

# INAUGURAL-DISSERTATION

zur

**Erlangung der Doktorwürde**

der

**Naturwissenschaftlich-Mathematischen Gesamtfakultät**

der

**Ruprecht-Karls-Universität**

**Heidelberg**

vorgelegt von

**Mag. Thomas Quirchmayr**

Heidelberg

Tag der mündlichen Prüfung: .....



# Retrospective Semi-automated Software Feature Extraction from Natural Language User Manuals

Gutachterin: Prof. Dr. Barbara Paech (Universität Heidelberg)  
Betreuer: Prof. Dr. Kurt Schneider (Universität Hannover)



# Abstract

Mature software systems comprise a vast number of heterogeneous system capabilities which are usually requested by different groups of stakeholders and evolve over time. Software features describe and logically bundle low level capabilities on an abstract level and thus provide a structured and comprehensive overview of the entire capabilities of a software system. Software features are often not explicitly managed. Quite the contrary, software feature-relevant information is often spread across several software engineering artifacts (e.g., user manual, issue tracking systems). It requires huge manual effort to (1) identify and extract software feature-relevant information from these artifacts in order to make software feature knowledge explicit and furthermore to (2) determine which software features the disclosed software feature-relevant information belongs to. This thesis presents a three-step-approach to semi-automatically enhance software features by software feature-relevant information from a user manual: first, a domain terminology is semi-automatically extracted from a natural language user manual based on linguistic patterns. Second, the extracted domain terminology, structural sentence information and natural language processing techniques are used to automatically identify and extract atomic software feature-relevant information with an  $F_1$ -score of at least 92.00%. Finally, the determined atomic software feature-relevant information is semi-automatically assigned to existing and logically related software features. The approach is empirically evaluated by means of a user manual and corresponding gold standards of an industrial partner. This thesis provides tool support to identify and extract atomic software feature-relevant information from user manuals and furthermore recommend logically related software features.

# Zusammenfassung

Softwaresysteme umfassen häufig eine umfangreiche Sammlung an heterogenen Systemeigenschaften. Diese werden typischerweise von unterschiedlichen Anwendergruppen gefordert und können sich im Laufe der Zeit ändern. Softwarefeatures beschreiben und bündeln diese heterogenen Systemeigenschaften und bieten daher einen strukturierten, detaillierten und umfassenden Überblick über die Eigenschaften eines Softwaresystems. Jedoch werden Informationen über Softwarefeatures meist nicht explizit verwaltet und beschrieben sondern sind oft über viele Software Engineering Artefakte verteilt (zB. Nutzerhandbücher).

Zusätzlich wird of großer Aufwand benötigt, um Softwarefeature-relevante Informationen (1) in Software Engineering Artefakten zu identifizieren und aus diesen zu extrahieren sowie (2) die den Informationen zugehörige Softwarefeatures zu bestimmen. In dieser Doktorarbeit wird ein dreistufiger Ansatz entwickelt, durch den Softwarefeatures semiautomatisch mit aus Nutzerhandbüchern extrahierten Softwarefeature-relevanten Informationen bereichert werden. Dafür wird zuerst aus dem Nutzerhandbuch eine Domänenterminologie mit Hilfe linguistischer Muster semiautomatisch extrahiert. Anschließend werden die extrahierte Domänenterminologie, strukturelle Satzinformationen und Techniken maschineller Sprachverarbeitung verwendet, um Softwarefeature-relevante Informationen automatisch in Nutzerhandbüchern zu identifizieren und daraus zu extrahieren. Schlussendlich werden die extrahierten Informationen semiautomatisch den logisch zugehörigen Softwarefeatures zugewiesen. Der gesamte in dieser Doktorarbeit vorgestellte Ansatz wird mit Hilfe eines Nutzerhandbuches und entsprechenden Goldstandards empirisch validiert und interpretiert. Im Zuge dieser Doktorarbeit wird ein Werkzeug entwickelt mit dessen Hilfe die Identifikation und Extraktion von Softwarefeature-relevanten Information aus Nutzerhandbüchern semiautomatisch möglich ist. Darüber hinaus unterstützt das Werkzeug die Zuweisung dieser Informationen zu den logisch zugehörigen Softwarefeatures.

# Acknowledgements

First and foremost, I would like to thank Prof. Dr. Barbara Paech for her supervision over the last four years, her support, continuous feedback, and valuable discussions. I am very grateful for the opportunity to work and research in her Software Engineering Group at the Faculty of Mathematics and Computer Science of Heidelberg University. Furthermore, I would like to thank my external supervisor Prof. Dr. Kurt Schneider for his helpful suggestions and his time for evaluating my thesis.

Second, I want to thank Roche Diagnostics GmbH in Mannheim for the great opportunity to work and research in their company. In specific, many thanks to Hannes Karey, Roland Kohl, and Gunar Kasdepke for their ideas, support, and the many (not only research-related) discussions.

I would also like to thank my colleagues at the Software Engineering Group for discussions, feedback, and in specific for the many other activities not related to work which helped to brighten up things a lot! Thanks to Anja Kleebaum for her amazing cake, Christian Kücherer for his home-grown vegetables, Marcus Seiler for his “Sächsisch”, Paul Hübner for his bike-related support, Thorsten Merten for his (far too rare) visits and related night-life experiences in Heidelberg, and Tom-Michael Hesse for knowing answers to literally everything! Additionally, many thanks to Doris Keidel-Müller and Willi Springer for their administrative support and open ears all over the years!

Finally, I want to especially thank my parents Michaela and Ernst, my brother Andreas, my sister in law, Christiane, and my half-year old niece, Livia: thanks for your love, support, advice, and great times all over the years! Last and most of all: infinite thanks to Birgit for your love, patience (yes, really!), support, advice, and the amazing four years together in Heidelberg!





# Contents

<b>I Preliminaries</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Problem Statement . . . . .	3
1.3 Contributions . . . . .	4
1.4 Structure of the Thesis . . . . .	5
1.5 Publications arising from this Thesis . . . . .	6
<b>2 Foundations</b>	<b>7</b>
2.1 Research Methodology . . . . .	7
2.2 User Manuals as Core Artifacts in Software Engineering . . . . .	8
2.3 Software Feature and Feature-relevant Information . . . . .	9
2.4 Domain-specific Terminology . . . . .	11
2.5 Natural Language Processing . . . . .	11
2.5.1 Morphological Analysis . . . . .	12
2.5.2 Lexical Analysis . . . . .	13
2.5.3 Syntactic Analysis . . . . .	14
2.6 Text Mining . . . . .	15
2.6.1 Text Preprocessing . . . . .	16
2.6.2 Text Representation . . . . .	17
2.6.3 Knowledge Discovery in Text . . . . .	17
2.6.3.1 Information Extraction . . . . .	18
2.6.3.2 Text Classification . . . . .	19
2.6.3.3 Text Clustering . . . . .	19
2.7 Measurement Fundamentals . . . . .	20
<b>II Problem Investigation</b>	<b>22</b>
<b>3 Software Feature Extraction from Natural Language Text: State of the Art</b>	<b>23</b>
3.1 Research Questions . . . . .	23
3.2 Review Method . . . . .	25

3.3	Results . . . . .	28
3.3.1	Findings in Selected Publications . . . . .	29
3.3.1.1	Which approaches exist to extract software feature-relevant information from natural language software engineering artifacts? . . . . .	30
3.3.1.2	Which software feature-relevant entities are extracted? Which types of natural language software engineering artifacts are mined? . . . . .	33
3.3.1.3	Which technologies are used? . . . . .	34
3.3.1.4	Which degree of automation do the approaches provide? . . . . .	35
3.3.1.5	Which supporting manual effort is needed to apply the approaches? . . . . .	36
3.4	Summary and Conclusion . . . . .	36
<b>III Treatment Design</b>		<b>37</b>
<b>4</b>	<b>Software Feature Extraction (SOFEX)</b>	<b>38</b>
4.1	Requirements . . . . .	38
4.2	Overview . . . . .	39
4.3	Information Identification (semi-automated) . . . . .	40
4.3.1	User Manual Revision (manual, optional) . . . . .	41
4.3.2	Document Preparation (automated) . . . . .	42
4.3.3	Terminology-related (TR) Preprocessing (automated) . . . . .	44
4.3.4	Terminology Extraction (semi-automated) . . . . .	44
4.3.4.1	Candidate Extraction (automated) . . . . .	45
4.3.4.2	Term Validation (semi-automated) . . . . .	46
4.3.5	Software Feature-relevant Sentence Identification (automated) . . . . .	47
4.4	Information Extraction (automated) . . . . .	49
4.4.1	Information-related (IR) Preprocessing (automated) . . . . .	50
4.4.1.1	Terminology-based textual modifications (automated) . . . . .	50
4.4.1.2	Pattern-based parse tree transformations (automated) . . . . .	51
4.4.2	Software Feature-relevant Information Extraction (automated) . . . . .	53
4.5	Information Assignment (semi-automated) . . . . .	57
4.5.1	Assignment-related (AR) Preprocessing (automated) . . . . .	57
4.5.2	Learning (automated) . . . . .	59
4.5.3	Software Feature Knowledge Enhancement (manual) . . . . .	60
<b>IV Treatment Validation</b>		<b>61</b>
<b>5</b>	<b>Evaluation</b>	<b>62</b>
5.1	Introduction . . . . .	62
5.2	Evaluation I . . . . .	65
5.2.1	Information Identification (Id) . . . . .	66
5.2.2	IR Preprocessing (Pp) . . . . .	69
5.2.3	Information Extraction (Ix) . . . . .	70

5.2.4	Information Assignment (As) . . . . .	71
5.2.5	Discussion . . . . .	74
5.3	Evaluation II . . . . .	76
5.3.1	Terminology Extraction (Tx) . . . . .	76
5.3.2	Information Identification (Id) . . . . .	80
5.3.3	IR Preprocessing (Pp) . . . . .	81
5.3.4	Information Extraction (Ix) . . . . .	82
5.3.5	Information Assignment (As) . . . . .	83
5.3.6	Discussion . . . . .	84
<b>V</b>	<b>Conclusion</b>	<b>86</b>
<b>6</b>	<b>Discussion</b>	<b>87</b>
6.1	Application Effort & Adaption Need . . . . .	87
6.1.1	Manual Effort to Apply SOFEX . . . . .	87
6.1.2	Manual Effort to Adapt SOFEX . . . . .	89
6.2	Threats to Validity . . . . .	90
6.2.1	Threats to Conclusion Validity . . . . .	90
6.2.2	Threats to Internal Validity . . . . .	91
6.2.3	Threats to Construct Validity . . . . .	91
6.2.4	Threats to External Validity . . . . .	91
<b>7</b>	<b>Summary</b>	<b>93</b>
<b>8</b>	<b>Future Work</b>	<b>95</b>
<b>VI</b>	<b>Appendix</b>	<b>97</b>
<b>A</b>	<b>Natural Language Processing</b>	<b>98</b>
A.1	Penn Tag Set . . . . .	98
A.2	Stopwords . . . . .	98
<b>B</b>	<b>Parse Tree Modification patterns</b>	<b>101</b>
B.1	Parse tree correction patterns . . . . .	101
B.1.1	JJ to NN . . . . .	101
B.1.2	ADVP to NP . . . . .	101
B.1.3	Cleanse PP . . . . .	102
B.1.4	VP to JJ . . . . .	102
B.1.5	ADJP to PP . . . . .	103
B.1.6	Complex NP#1 . . . . .	103
B.1.7	Complex NP#2 . . . . .	103
B.1.8	Complex NP#3 . . . . .	104
B.1.9	Complex NP#4 . . . . .	104
B.1.10	Complex NP#5 . . . . .	104
B.1.11	Cleanse PP . . . . .	105
B.1.12	Cleanse NP lists#1 . . . . .	105

B.1.13	Cleanse NP lists#2 . . . . .	106
B.1.14	Cleanse S#1 . . . . .	106
B.1.15	Cleanse S#2 . . . . .	106
B.1.16	Cleanse "between" #1 . . . . .	107
B.1.17	Cleanse "between" #2 . . . . .	108
B.2	Parse tree adaption patterns . . . . .	110
B.2.1	Remove SINV . . . . .	110
B.2.2	Remove Brackets . . . . .	110
B.2.3	Cleanse FRAG . . . . .	110
B.2.4	Complex VP#1 . . . . .	111
B.2.5	Complex VP#2 . . . . .	111
B.2.6	Complex VP#3 . . . . .	111
B.2.7	Complex VP#4 . . . . .	112
B.2.8	ADVP in VP#1 . . . . .	112
B.2.9	ADVP in VP#2 . . . . .	112
B.2.10	ADJP in VP . . . . .	113
B.2.11	PRT in VP . . . . .	113
B.2.12	Complex PP . . . . .	113
B.2.13	Complex NP#6 . . . . .	114
B.2.14	Multiple PP#1 . . . . .	114
B.2.15	Multiple PP#2 . . . . .	115
B.2.16	Remove S#1 . . . . .	115
B.2.17	Remove S#2 . . . . .	116
B.2.18	SBAR to VPH . . . . .	116
B.2.19	SBAR to VPC#1 . . . . .	117
B.2.20	SBAR to VPC#2 . . . . .	117
B.2.21	SBAR to VPP . . . . .	118
B.2.22	VP to VPV . . . . .	118
B.2.23	PP to VPP#1 . . . . .	118
B.2.24	PP to VPP#2 . . . . .	119
B.2.25	VP to VPT#1 . . . . .	119
B.2.26	VP to VPT#2 . . . . .	119
B.2.27	VP to VPT#3 . . . . .	120
B.2.28	VP to VPW . . . . .	120
B.2.29	PP to NPP . . . . .	120
B.2.30	SBAR to NPW . . . . .	121
B.2.31	VP to NPV . . . . .	121
B.2.32	NP to PPN . . . . .	122
B.2.33	PP to PPV . . . . .	122
B.2.34	PP to PPW#1 . . . . .	122
B.2.35	PP to PPW#2 . . . . .	123
B.2.36	Surround NP . . . . .	124

# List of Figures

1.1	Problem Space . . . . .	4
1.2	Thesis Structure . . . . .	5
2.1	Engineering Cycle . . . . .	8
2.2	Linguistic Information Model (LIM) . . . . .	10
2.3	LIM Components based on 2 example sentences . . . . .	10
2.4	Morphological Terminology and Example . . . . .	12
2.5	Syntactic Analysis . . . . .	15
2.6	Text Mining Framework . . . . .	16
3.1	Software Feature-relevant Information Extraction Characteristics . . . . .	25
3.2	Process of the Literature Review . . . . .	28
3.3	Publication Chronology . . . . .	29
4.1	SoFeX Process Steps . . . . .	40
4.2	Semi-automated Information Identification . . . . .	41
4.3	Document Preparation (left: user manual excerpt, right: <i>Dataset</i> ) . . . . .	42
4.4	Sentence Type Examples . . . . .	43
4.5	Terminology Extraction Process . . . . .	45
4.6	Different Types of Terminology Candidate Validation . . . . .	47
4.7	Information Filtering Example . . . . .	48
4.8	Information Extraction . . . . .	49
4.9	Parse tree accuracy increases with domain term ( <b>bold</b> ) bundling . . . . .	50
4.10	Parse Tree Transformation Example in Java . . . . .	51
4.11	Condense Complex Noun Phrases in Parse Trees . . . . .	52
4.12	Condense Complex Verb Phrases in Parse Trees . . . . .	52
4.13	Indicate LIM Complements and Modifiers in Parse Trees . . . . .	53
4.14	Information Simplification Example . . . . .	55
4.15	Enumeration Resolution Example . . . . .	55
4.16	Information Assignment Process . . . . .	57
4.17	Feature Hierarchy and Feature Description Meta Model . . . . .	58
4.18	Roche's GDC Feature Document Transformation . . . . .	58
4.19	Exemplary Classification Model (Weka ARFF File) . . . . .	59
4.20	Feature Knowledge Enhancement Process . . . . .	60

5.1	SoFeX Evaluation Overview . . . . .	63
5.2	Evaluations Overview (Gold Standards) . . . . .	66
5.3	Evaluation Overview (Domain Experts) . . . . .	77

# List of Tables

1.1	Scientific Publications in Context of the Thesis . . . . .	6
2.1	Tokenizer Results . . . . .	14
3.1	PICOC criteria . . . . .	24
3.2	Research Questions for the SLR . . . . .	24
3.3	Definition of search terms . . . . .	26
3.4	Inclusion (I#) and Exclusion Criteria (E#) . . . . .	27
3.5	Journals and Conferences for manual target search (alphabetically ordered)	28
3.6	Publication Venues of the selected Studies . . . . .	30
3.7	Characteristics of the SLR Publications . . . . .	30
3.8	Types of EXT and FRI . . . . .	33
3.9	NLP Characteristics of the SLR Publications . . . . .	34
4.1	Syntactical Relevancy Patterns . . . . .	56
5.1	Different Evaluations of SOFEX . . . . .	64
5.2	Evaluation Results for Information Identification . . . . .	67
5.3	Evaluation of Information Identification with Clustering and Classification	68
5.4	Parse Tree Accuracy for IR Preprocessing . . . . .	70
5.5	Evaluation Results for Information Extraction . . . . .	71
5.6	Evaluation Results for Information Assignment . . . . .	73
5.7	Evaluation Results for Domain Terminology Extraction . . . . .	79
5.8	Evaluation Results for Information Identification . . . . .	80
5.9	Parse Tree Accuracy for IR Preprocessing . . . . .	81
5.10	Evaluation Results for Information Extraction . . . . .	82
5.11	Evaluation Results for Information Assignment . . . . .	83
6.1	Manual Effort to Apply SOFEX (in hours) . . . . .	88
A.1	Penn Tag Set . . . . .	99
A.2	Stopword List . . . . .	100

# List of Abbreviations

BOW	Bag-of-Words
CRM	Customer Relationship Management
DM	Data Mining
EM	Expectation-Maximization
FN	False Negative
FP	False Positive
GDC	Global Deal Calculator
IE	Information Extraction
KM	K-Means
LIM	Linguistic Information Model
ML	Machine Learning
NB	Naive Bayes
NL	Natural Language
NLP	Natural Language Processing
POS	Part-of-Speech
RE	Requirements Engineering
RQ	Research Question
SDLC	Software Development Lifecycle
SE	Software Engineering
SLR	Systematic Literature Review
SoFEX	Software Feature Extraction Approach
SVM	Support Vector Machine
TM	Text Mining
TN	True Negative
TP	True Positive
VSM	Vector Space Model



# Part I Preliminaries

# Introduction

## 1.1 Motivation

Mature software systems comprise a vast number of heterogeneous system capabilities which evolve over time and are usually requested by different groups of stakeholders (e.g., Godfrey and German, 2008). This diversity of stakeholders together with requirements for software systems, and primarily software system evolution, often result in several unstructured, incomplete and inconsistent descriptions of the software system capabilities (Forward and Lethbridge, 2002). Software system capabilities are initially described in requirements engineering artifacts (e.g., use case diagram, activity diagram, process models, requirement specification, etc.). Software systems evolve in order to adapt in a timely manner to their changing environment and to meet stakeholder needs (Godfrey and German, 2008). This evolution requires to revise corresponding software engineering artifacts - code as well as documents - to provide up-to-date information about the systems capabilities and, of course, a software system which fulfills all the stakeholder needs in time.

A proper framework to describe system capabilities in a structured way are software features and corresponding feature models. The necessity of feature-based software system descriptions is manifold: release planning in software product management (see, e.g., Zorn-Pauli et al., 2012), software product line engineering (see, e.g., Bakar et al., 2015), requirements feature interaction detection (see, e.g., Shaker et al., 2012), stakeholder communication (see, e.g., Pikkarainen et al., 2008), as well as software product comparison (see, e.g., Earls et al., 2002).

However, software features are often not explicitly managed, at least not from scratch. Rather, software feature-relevant information is spread across several software engineering (SE) artifacts. Therefore, these artifacts need to be searched in order to uncover software

feature-relevant information. Paech et al. (2014) pointed out that user manuals can serve as an appropriate source to gather software feature-relevant information from a user workflow-driven perspective. Depending on the size of the software system and thus the size of the corresponding user manual, manual software feature-relevant information extraction is cumbersome, error-prone, and costly (see, e.g., Weston et al. (2009)). As a consequence, our overall research goal is to provide automated support to extract software features and related atomic software feature-relevant information from natural language user manuals.

This thesis was developed in cooperation with Roche Diagnostics GmbH (Roche in the following). Recently, Roche uses software features to support the process of software engineering for a bespoke in-house customer relationship management (CRM) software called Global Deal Calculator (GDC) which supports contract life cycle management. But, there does not exist an explicit description of the entire software features of GDC. Thus, Roche seeks for an approach to gather software feature-relevant information from available corresponding software engineering artifacts in a semi-automated way. Previously, Paech et al. (2014) showed that user manuals are in general a valuable source for extracting software feature information. Hence, this thesis proposes a new semi-automated approach to gather software feature-relevant information from a user manual to generate an overall software feature-based system description.

## 1.2 Problem Statement

Often, software features and software feature-relevant information is not explicitly managed from the very beginning of a software development lifecycle (SDLC). Thus, software feature-relevant information needs to be uncovered and collected retrospectively.

The SDLC comprises a number of different work phases which depend on the used SDLC model, like waterfall or agile (Bourque and Fairley, 2014). Typically and independent of the used SDLC model, each SDLC must comprise some key phases which are necessary to successfully build software systems (see Figure 1.1). In each phase several natural language software engineering artifacts (NLSEA) might be created, maintained and revised. These NLSEA, especially user manuals (Paech et al., 2014), contain software feature-relevant information.

This thesis presents an approach, called SOFEX, to semi-automatically identify and extract atomic software feature-relevant information from natural language user manuals as well as the enhancement of existing software features with the corresponding software feature-relevant information. Thus, this thesis targets three primary goals related to the enhancement of software features:

- G1 Design an approach to identify software feature-relevant sentences in unconstrained and unstructured natural language user manuals.
- G2 Design an approach to extract atomic software feature-relevant information from software feature-relevant sentences.
- G3 Design an approach to enrich existing software features with software feature-relevant information.

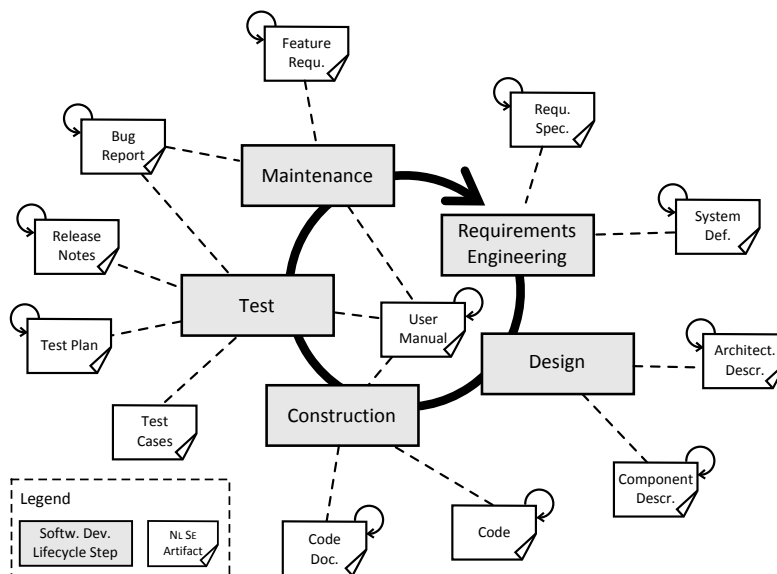


Figure 1.1: Problem Space

### 1.3 Contributions

This thesis provides three contributions out of knowledge on software feature extraction from natural language software engineering artifacts. First, a systematic mapping study (systematic literature review, SLR) is used to provide an overview of the state-of-the-art and -practice for software feature-relevant information extraction from natural language software engineering artifacts. Second, based on the findings of the SLR, SoFEX is designed comprising three subsequent core steps:

- semi-automatic extraction of a domain-specific terminology from natural language text
- automatic identification and extraction of atomic software feature-relevant information based on a domain-specific terminology and linguistic patterns
- semi-automatic enhancement of existing software features with the extracted atomic software feature-relevant information

Third, SoFEX is empirically evaluated based on a natural language user manual from Roche which describes a mature CRM software system.

## 1.4 Structure of the Thesis

This thesis is structured in five parts and six chapters with parts II-IV structured along the design science methodology approach (Wieringa and Morali, 2012, see Section 2.1). Figure 1.2 shows an overview of the thesis structure and the corresponding research questions and results.

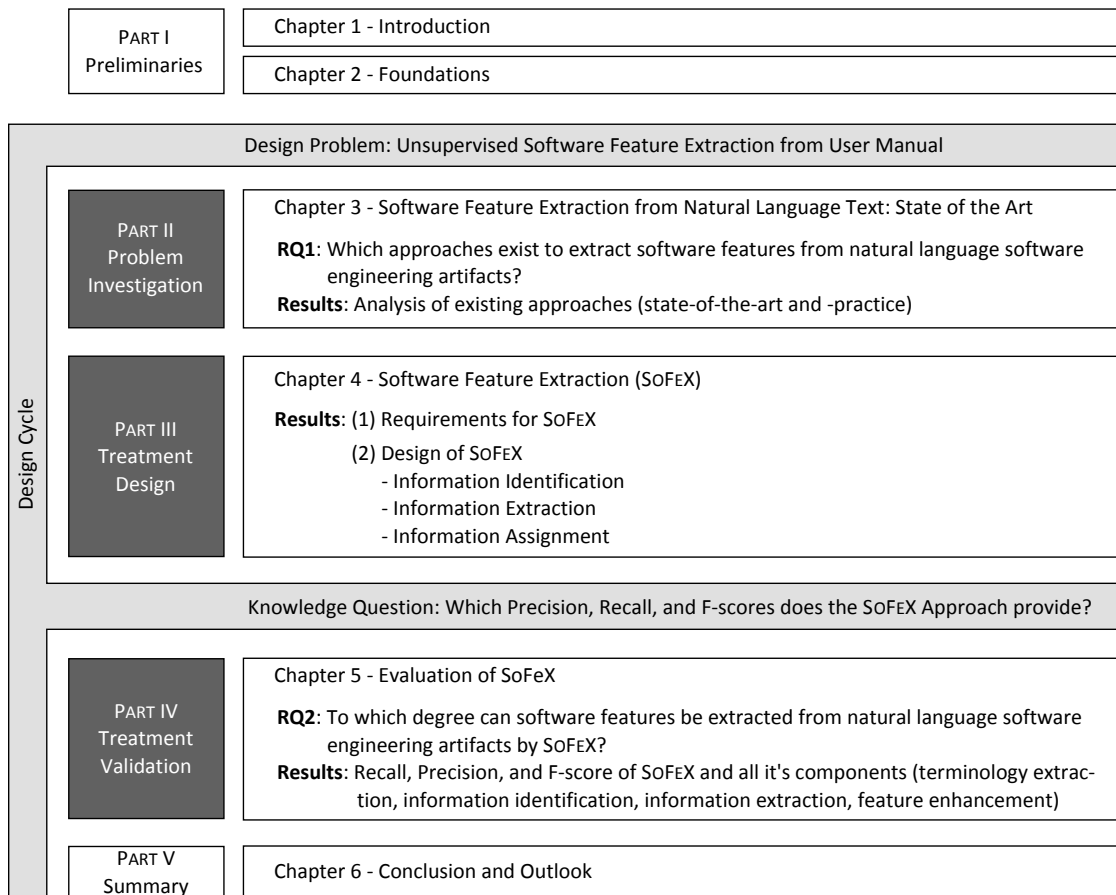


Figure 1.2: Thesis Structure

Part I (Preliminaries) introduces the motivation for the thesis, describes related problems, and contributions of the thesis (Chapter 1). Furthermore, the research methodology and important foundations are included in Chapter 2.

Part II (Problem Investigation) investigates the problem and presents the results of a systematic mapping study regarding state-of-the-art and -practice regarding software feature extraction from natural language software engineering artifacts (Chapter 3).

Part III comprises a detailed overview of SoFEX' requirements and a detailed description of SoFEX taking the goals and requirements into account (Chapter 4).

Part IV presents the evaluation of SoFEX which consists of several separate evaluations according to the the components of SoFEX (preprocessing, terminology extraction, information identification, information extraction, and information enhancement).

## 1.5 Publications arising from this Thesis

Parts of this thesis have already been published as scientific work. Table 1.1 provides an overview of these publications in chronological order and the corresponding chapter.

Table 1.1: Scientific Publications in Context of the Thesis

---

No	Publication	Chapter
1	T. Quirchmayr, B. Paech, H. Karey, R. Kohl: <b>Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals</b> , In: Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17), Essen, March 2017	4, 5
2	T. Quirchmayr, B. Paech, H. Karey, R. Kohl, G. Kasdepke: <b>Semi-automatic Rule-based Domain Terminology and Software Feature-relevant Information Extraction from Natural Language User Manuals</b> . Empirical Software Engineering X(Y), 2018	4, 5

---

# Chapter 2

## Foundations

This chapter provides definitions of the most important terms and concepts used in this thesis. Sections 2.1 and Sections 2.2 briefly introduce the research methodology and user manuals as a core artifact in software engineering, respectively. Section 2.3 describes software features, software feature-relevant information, as well as the linguistic information model which is used to capture software feature-relevant information. Section 2.4 shortly introduces the concepts of domain term and domain-specific terminology while Section 2.5 describes natural language processing in detail. Section 2.6 explains the process of text mining and applications. Finally, Section 2.7 gives insights into the performance measures used for evaluation purpose.

### 2.1 Research Methodology

This thesis follows the **Design Science** methodology from Wieringa and Morali (2012). Design science was first introduced by Fuller (1957) as a ”*systematic form of designing*“. In detail, Wieringa and Morali (2012) extended the term design science in the context of software engineering as ”*design and investigation of artifacts [...] to interact with a problem context in order to improve something in the context*“ and empirical evaluation of the artifact. An artifact is something that is created for a practical purpose, the interaction between an artifact and its context is called treatment.

Design science distinguishes between the two problem-solving cycles design cycle and empirical cycle. The **design cycle** is used to design and investigate artifacts to solve stakeholder’s design problems (e.g., design a software feature extraction approach). The **empirical cycle** aims to answer (empirical) knowledge questions about an artifact within its context (e.g., which accuracy does the software feature extraction approach provide in practice?). Both, the design cycle and the empirical cycle are part of the **engineering**

**cycle.** Figure 2.1 shows the engineering cycle which consists of the following five tasks:

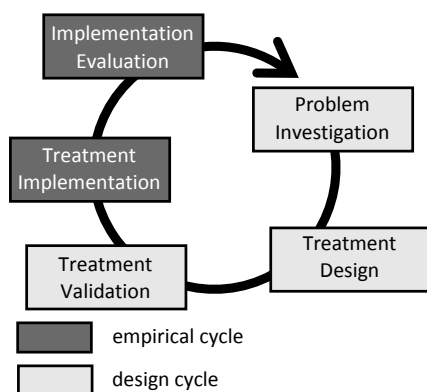


Figure 2.1: Engineering Cycle

Within the first task, named Problem Investigation, information is collected about the relevant problem in order to prepare for the subsequent task Treatment Design. The second task, Treatment Design, is related to the design of an artifact (decision about what to do) and its documentation (e.g., specification). The aim of the third task, Treatment Validation, is to predict how a treatment performs in comparison to its intended use. The application of a treatment to the original problem is called Treatment Implementation

(fourth task). Finally, in contrast to treatment validation, the goal of the last task, Implementation Evaluation, is to investigate how an artifact interacts in a real-world context (e.g., case study). In context of the thesis, only the design cycle (and thus the first three steps of the engineering cycle) is applied, because SOFEX is not implemented and evaluated in a real-world context by domain experts.

## 2.2 User Manuals as Core Artifacts in Software Engineering

Paech et al. (2014) already showed, that user manuals serve as an appropriate source to gather software feature-relevant information from a user workflow-driven perspective. However, gathering and extracting information from natural language documents poses the challenge, that they usually contain unstructured or semi-structured content which is not machine-readable. More precisely, these documents lack semantic meta data (Aggarwal and Zhai, 2012a). In context of the thesis, the GDC user manual was chosen to serve as primary source to extract software feature-relevant information for the following reasons:

- The GDC user manual is aligned with the business processes within contract life cycle management from a business user perspective. Thus, the user manual contains all the information necessary for a business user to use GDC in its entire range.
- The GDC user manual is maintained by a single person. Thus, the manual reuses consistent domain-specific phrases, which indicate the location of software feature-relevant information.
- The GDC user manual is kept up-to-date, mature and comprehensive.



## 2.3 Software Feature and Feature-relevant Information

The term *feature* is widely used in computing: from image processing (e.g., image structure in Nixon (2008)), signal processing (e.g., aiming to capture specific aspects of audio signals in a numeric way in Nixon (2008)) to computer linguistics (e.g., property of a class of linguistic terms which describes individual members of this class in Corbett (2006)), machine learning (e.g., specification of an attribute and its value in Bishop (2006)), and software engineering (e.g., characteristic of a software item in IEEE (1990)). Even within the domain of software engineering, there is neither a common understanding nor a precise definition of a feature (see, e.g., Apel and Kästner, 2009; Classen et al., 2008; Marcuska et al., 2014). Therefore, in context of this thesis, a feature is defined as follows (inspired by Bosch (2000) and Eisenbarth et al. (2003)):

A **software feature** describes an abstract unit of behaviour of a software system at a high level and bundles atomic information which describe the unit of behaviour at a detailed level. The latter is called **atomic unit of software feature-relevant information**.

As an example, “*Quantificator calculates materials*” is an atomic software feature-relevant unit of information of the feature “*Price Calculation*”. An atomic unit of information, in contrast to combined information, cannot be broken down into other simpler units of information without losing information (see, e.g., Chandrasekar et al., 1996; Jonnalagadda et al., 2009). In short, the aim is to extract smallest bits of information. Based on the requirement of information *atomicity*, a **linguistic information model** (LIM) is defined in order to capture atomic software feature-relevant information (see Figure 2.2) on a syntactic level. The LIM simplifies (e.g., *adjective phrases* are not explicitly considered but are part of the *noun phrase*) as well as extends (e.g, *if clauses* are added) the English phrasal structure (see, e.g., Brinton and Brinton, 2010). The LIM is customized in order to finally determine if a potentially atomic unit of software feature-relevant information is truly software feature-relevant based on it’s syntax (see Section 4.4).

An atomic unit of *information* is a *clause*. A *clause* is defined as grammatical structure which contains a *subject* and a *predicate*. It is either an *information*, a *to clause* (e.g., start *to run*), a *conditional clause* (e.g., start *if...*), a *that clause* (e.g., show *that...*), or a *wh clause* (e.g., the material *which...*). Additionally, clauses comprise a *clause term* and an *information*. In contrast to other clauses, a conditional clause might contain more than one atomic unit of information as combined conditions (e.g., *If the quantificator runs AND the error is shown*) that cannot be separated without losing information. A *predicate* is a phrase that expresses the action performed by the subject. It requires a

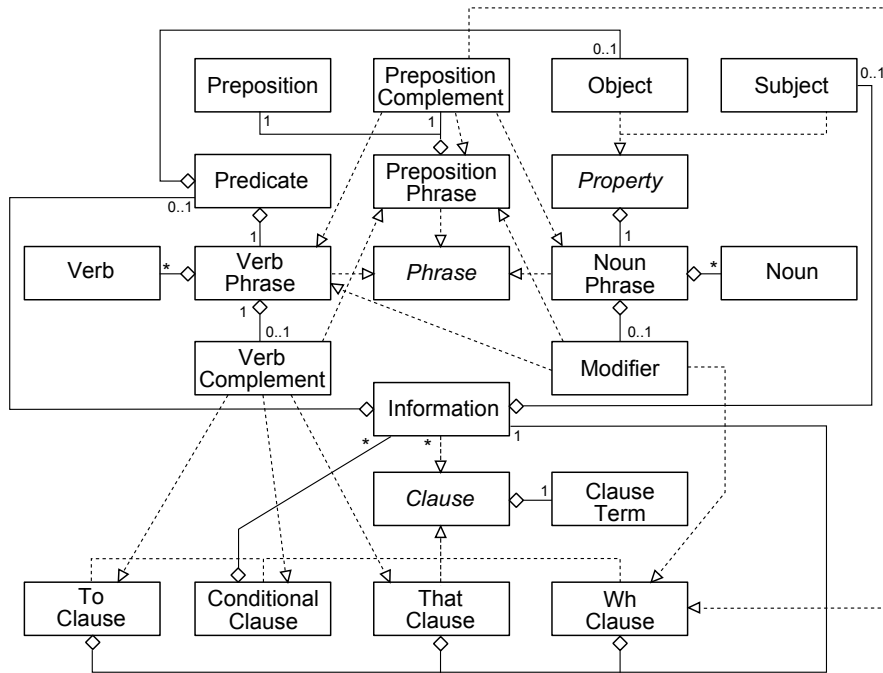


Figure 2.2: Linguistic Information Model (LIM)

*verb phrase* which comprises at least one *verb* and optional *verb complements*. A verb complement is a phrase (preposition) or clause (to, conditional, that) that completes the meaning of a verb phrase. Furthermore, a *predicate* might include an *object*. Both *subject* and *object* are a *property*, which contains a *noun phrase*. A *noun phrase* consists of *nouns* and *modifiers*. A noun phrase modifier is a phrase (preposition, verb) or a wh clause that modifies or describes a noun phrase. For simplification, adverbs and adjectives are included as part of *verbs* and *nouns*. Figure 2.3 depicts some components of the LIM based on two example sentences.

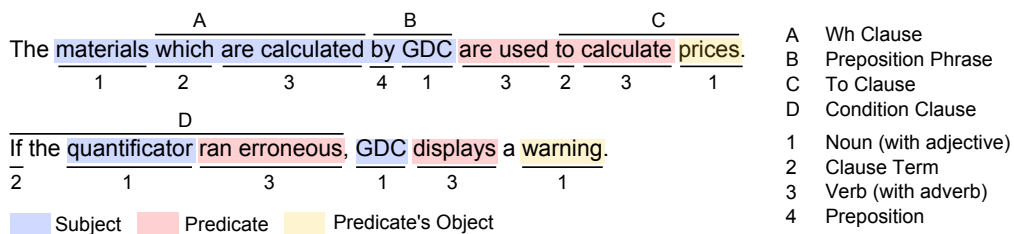


Figure 2.3: LIM Components based on 2 example sentences

## 2.4 Domain-specific Terminology

A domain-specific terminology (domain terminology) corresponds to a set of reusable domain-specific terms (domain term). Each domain term has a significant meaning (Kim and Cavedon, 2011) and represents a key concept which exists in and thus describes a given domain of interest (Castañeda et al., 2010). A domain term can be interpreted without any ambiguity (Hsieh et al., 2011). Additionally, domain terms are closed to objects (physical, e.g., "material", or logical, e.g., "parameter"; Noy and McGuinness, 2001) or processes (e.g., "customization"), in contrast to non-domain terms (e.g., "step").

In context of SOFEX, the following definition for domain term and domain terminology is used:

A **domain term** (DT) is a lexical realization of something important or relevant to a domain. A domain term can be simple (single word) or complex (multi word). It comprises at least one noun and might contain adjectives, verbs, and/or adverbs. A set of domain terms of a specific domain is called **domain terminology**.

A single word is also called **unigram**. Complex words which consist of two words are called **bigrams**, **trigrams** are three-word terms.

## 2.5 Natural Language Processing

Natural language processing (NLP) is an area of research within computer linguistics. Its purpose is to enable computers to understand and manipulate natural language text or speech. Chowdhury (2003) and lidd98 define NLP as follows:

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

Natural language processing follows the human process of natural language (NL) understanding. Humans usually follow several independent levels in order to understand natural languages and thus decode meaning from text and speech. Following Feldman (1999) and Liddy (1998), seven independent levels exist that humans follow in order to extract meaning from text or spoken language. On the **phonetical** level, humans analyse speech on its surface considering articulation, perception, as well as acoustic features. Second, on the **morphological** level humans determine the internal structure of words followed by a **lexical** analysis, which deals with lexical meaning of words and

parts of the speech. Several authors (e.g, Jurafsky and Martin (2014)) consider lexical analysis as part of the semantic analysis. Within the **syntactic** analysis the grammatical structure of a sentence is uncovered. After discovering the grammatical structure, possible meanings of sentences are determined (**semantic** level) followed by the interpretation in a larger context (**discourse** level). Finally, the purposeful use of a language within specific situations is applied in order to accomplish goals (**pragmatic** level).

NLP systems might incorporate all different levels or only some. The focus of this thesis is on morphological, lexical, syntactical, and semantic level which are explained in detail in the following sections.

### 2.5.1 Morphological Analysis

Morphology is one of the key fields in linguistics. It investigates the uncovering of the internal structure and formation of words which forms the basis for any NLP-based analysis. Based on the examples in Figure 2.4, the terminology which is necessary in context of the thesis and related to morphology, is introduced.

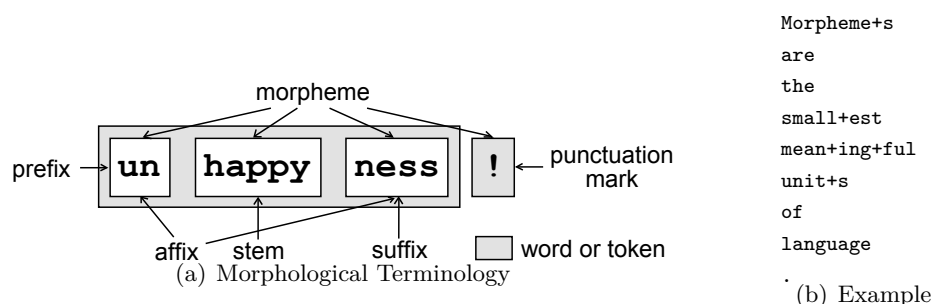


Figure 2.4: Morphological Terminology and Example

Figure 2.4(a) shows the morphological structure of an exemplary phrase "unhappiness!". It consists of four morphemes: a stem (or base, "happy"), two affixes ("un" - prefix, "ness" - suffix), and a punctuation mark ("!"). Thus, a morpheme is defined according to Bender (2013) as follows:

A **morpheme** is the smallest meaningful unit of language, usually consisting of a sequence of characters paired with concrete meaning. A morpheme may or may not stand alone.

A morpheme is not equal with a word: a word might be decomposable and thus consist of several morphemes, but an atomic word corresponds to a morpheme. In this thesis, a word is defined according to Bender (2013) as follows:

A **word** consists of one root (stem) morpheme and zero or more affixes. A word can stand alone.

Figure 2.4(b) shows a sentence which is split into words and morphemes; each line corresponds to a word, the morphemes are separated by ”+“ (e.g., the noun **morpheme** and its plural suffix ”s“). The words ”are“, ”the“, ”of“, and ”language“ are atomic words and are thus words and morphemes simultaneously. On the other hand, the word ”Morphemes“ consists of two morphemes.

Two complementary text segmentation (see, e.g., Manning et al., 2014) technologies are used in the thesis (see Section 2.6.1) which are attributed to the morphological analysis: stemming and lemmatization. **Stemming** is the process of stripping off affixes from a word in order to determine its stem (e.g., ”happy“ from Figure 2.4(a)). **Lemmatization** reduces inflected forms of canonical representations (infinitive for verbs, e.g., ”sang“, ”sung“, ”sings“) to it’s lemma or root (e.g., ”sing“).

### 2.5.2 Lexical Analysis

Lexical analysis aims to determine if a specific word or token belongs to or exists in a specific language. Therefore, lexical analysis aims to break down a text document into single sentences and words (Frakes and Baeza-Yates, 1992). In context of this thesis, two text segmentation technologies are used which belong to the lexical analysis, namely sentence boundary detection and tokenization. **Sentence boundary detection** (sentence splitting, sentence boundary disambiguation) aims to determine sentences which consist of one or more words in a text. Sentences end with punctuation (period, question mark, exclamation mark, quotation mark) (see, e.g., Grefenstette and Tapanainen, 1994; Hancke et al., 2012). Some punctuation marks can be almost unambiguous (e.g., ”!“, ”?“). Others (e.g., ”.“), are extremely ambiguous: it is, e.g., not trivial whether it is a full-stop or part of an abbreviation.

On the other hand, the aim of **tokenization** is to locate the token boundaries (e.g., whitespace, punctuation) and determine (word and non-word) tokens in a sequence of characters of a text. Tokenization in context of NLP may seem simple in a language that separates words or sentences by a special character (e.g., whitespace). However, not every language uses a distinct separation character (e.g. Chinese, Japanese, Thai). Even for languages (e.g., English) which use whitespace separation, a simple ”whitespace“-tokenizer is not sufficient to tokenize words due to several obstacles: one of these obstacles are hyphenated words (e.g., ”forty-one“), which can be interpreted as being either one word or two separate words (see, e.g., Grefenstette and Tapanainen, 1994). Table 2.1 shows that each tokenizer delivers slightly different results. This thesis used Stanford

NLP for the implementation of SoFEX.

Table 2.1: Tokenizer Results

No	Naive Whitespace	Apache Open NLP	Stanford NLP	Ideal
1	I	I	I	I
2	said,	said	said	said
3		,	,	,
4	'I	'I	,	,
5			I	I
6	can't	ca	ca	can
7		n't	n't	not
8	afford			afford
9	to	to	to	to
10	do	do	do	do
11	that.'	that	that	that
12		.	.	.
13		'	'	'

Tokenized sentence: *I said, 'I can't afford to do that.'*

### 2.5.3 Syntactic Analysis

In context of syntactic analysis, syntactic categories are assigned to each token in a sentence. Syntactic categories are known as **Part-of-Speech Tags** (PoS-tags, e.g., determiner, adjective, subject). In literature, there is no common agreement whether part-of-speech (PoS) analysis is part of morphological analysis, syntactical analysis, or forms an independent category of analysis ("Lexical Analysis"; see, e.g., Feldman (1999); Liddy (1998)). In this thesis, PoS analysis is described in the context of the syntactic analysis. Figure 2.5(a) shows an example sentence which is syntactically analysed by means of the Stanford NLP API (see, e.g., Manning et al. (2014)). In that context, the Stanford NLP PoS-Tagger (Toutanova and Manning, 2000; Toutanova et al., 2003) assigns corresponding PoS-tags to each word in the sentence (e.g., "morphemes" is assigned NN which refers to a plural noun in Figure 2.5(b)). For a detailed list of PoS-tags relevant for this thesis see Section A.1 in the Annex.

The key process of syntactic analysis is called **PoS Parsing**. PoS Parsing builds upon the PoS-tagged tokens and aims to generate a structural description of a sentence. More detailed, the linear and flat sequence of words is parsed and each word is converted into a hierarchical structure (**parse tree**). The parse tree shows how the words relate to each other (syntax). Basically, syntactic processing systems use a declarative representation of syntactic facts about a language (**grammar**). The PoS parser then compares that grammar with the sentence to be analysed to produce the sentence's parse tree (see, e.g.,

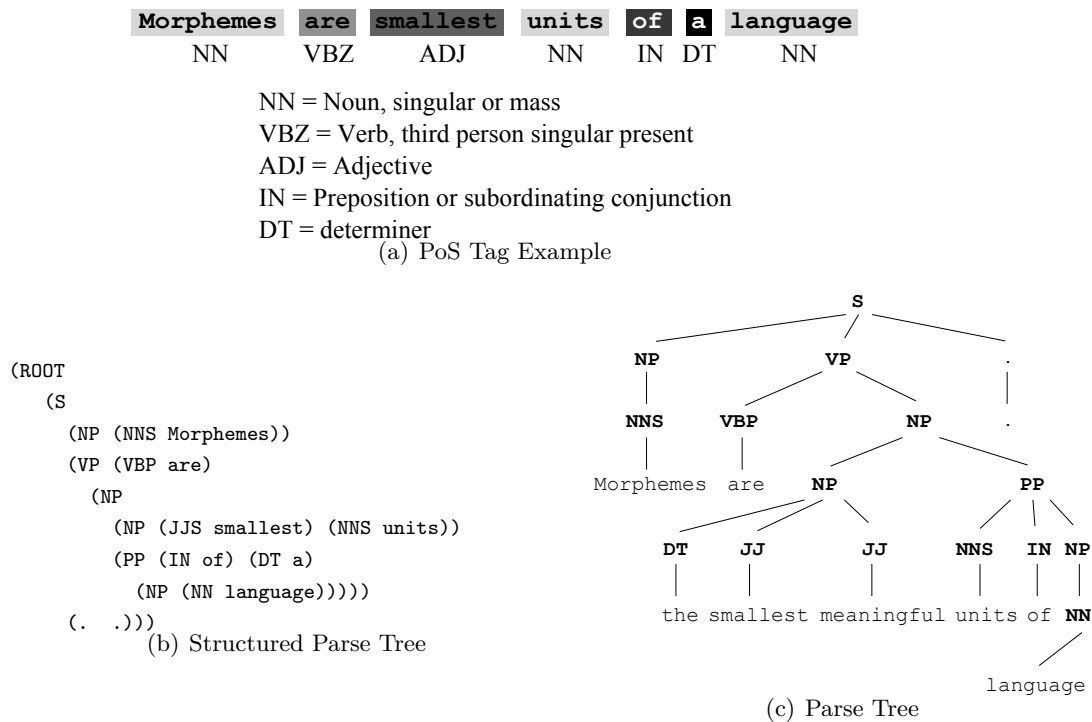


Figure 2.5: Syntactic Analysis

Dale et al., 2000; Jurafsky and Martin, 2014). Figures 2.5(b) and 2.5(c) show the parse tree based on the PoS tags from the exemplary sentence in Figure 2.4(b) in different representations. The upper part of the parse trees can be interpreted as "a sentence **S** is composed of a noun phrase **NP** followed by a verb phrase **VP** and a punctuation".

In context of the thesis, sentence simplification (see, e.g., Jonnalagadda et al., 2009) is used. **Sentence simplification** seeks to reduce the complexity of sentences. SOFEX utilizes a rule-based approach based on syntactic patterns (see Section 4.4). As an example, a sentence which contains a conjunction (e.g., and) is split into two atomic sentences: "The quantifier calculates materials and batch sizes." - (1) "The quantifier calculates materials." and (2) "The quantifier calculates batch sizes."

## 2.6 Text Mining

It is predicted that the data volume will grow to 40 billion terabytes by 2020, which equals a 50-time growth compared to the beginning of 2010 (Gantz and Reinsel, 2012). This tremendously increasing amount of data requires machine-aided technologies to determine relevant domain-specific information - data mining (DM) in general. Aggarwal (2015) defined data mining as "[...] *the automatic collection, cleansing, processing, and*

*analysis to get useful insights from data*“. **Text mining** (TM, text analytics) in specific, aims to derive high-quality information from textual data (e.g., patient records, health care insurance data, software engineering artifacts, etc.). Textual data can be structured (e.g., relational database), semi-structured (e.g., XML, JSON), or unstructured (e.g., word document, image). The term text mining was first introduced by Feldman and Dagan (1995). TM covers a broad set of related topics (e.g., DM) and techniques (e.g., sentiment analysis) to mine text (Aggarwal and Zhai, 2012a).



Figure 2.6: Text Mining Framework

Aggarwal and Zhai (2012a) identified three consecutive phases in a general framework for text mining, namely text preprocessing, text representation, and knowledge discovery in text (see Figure 2.6). Other authors (see, e.g., Hotho et al., 2005; Rajman and Besançon, 1998; Vijayarani et al., 2015) subsume text preprocessing (e.g., tokenization) and text representation (e.g., bag-of-words). The following sections introduce text mining techniques along the three phases text preprocessing, text representation, and knowledge discovery.

### 2.6.1 Text Preprocessing

Text preprocessing is one of the key components in many text mining algorithms. It aims at eliminating everything from text other than (preferably normalized) information-containing words (Aggarwal and Zhai, 2012a). At least in the context of text classification it was empirically evaluated that text preprocessing may have noticeable influence on the success of a text classification process (Uysal and Gunal, 2014). Typical preprocessing techniques comprise techniques from the morphological and syntactic analysis (see Sections 2.5.1 and 2.5.3): sentence detection, tokenization, filtering (stopword removal), lemmatization, and stemming (see, e.g., Aggarwal and Zhai, 2012a; Hotho et al., 2005) as well as PoS Tagging and Parsing (Hotho et al., 2005). Filtering aims to reduce the amount of text to be further processed. In context of text mining, **stopword removal** is most often used. Stopwords are very frequent and common words which appear to be of little contextual meaning and are thus not discriminative (see, e.g., Aggarwal and Zhai, 2012a; Jurafsky and Martin, 2014). Additionally, **lowercasing** is used to equalize different words by converting every character to lowercase. Finally, in order to increase the precision of the PoS parser (see Sections 4.3.3 and 4.4.1), string encoding (Ghosh et al., 2014) is used for domain terms and quoted phrases (phrases between quotation



marks). Therefore, the terms and phrases are bundled with delimiters and equipped with a prefix.

### 2.6.2 Text Representation

Natural language text can be represented by means of sparse numeric vectors. These vectors can be used to represent text as "Vector Space Model" or "Bag of Words Model". A **Bag-of-Words** (BOW) describes a piece of text (PoT)  $t_i$  by means of a disordered list of the words  $W_{t_i} = (w_{1,t_i}, \dots, w_{n,t_i})$  contained in the PoT (see, e.g., Jurafsky and Martin, 2014). A piece of text can be a document, paragraph, sentence, etc. For example,  $t_i = "a\ man\ saw\ a\ man"$  may result in a BoW  $b_{t_i} = \{'a', 'saw', 'man', 'a', 'man'\}$ . A **Vector Space Model** (VSM) represents a collection of PoTs as numerical vectors in an  $m$ -dimensional space (Hotho et al., 2005). Each PoT in the vector space is described by a numerical feature vector  $\vec{v}(t_i) = (o(t_i, w_1), \dots, o(t_i, w_m)) : w_j \in b_{t_i}$ . Each element of the vector usually represents a distinct word  $(w_1, \dots, w_m)$  of the PoTs BOW  $b_{t_i}$ . The size (or weight) of the vector is usually a function of the word frequency along with other factors (e.g., inverse document frequency) (see, e.g., Jurafsky and Martin, 2014). For example, three distinct words (with their occurrence) can be determined in  $b_{t_i}$ : a(2), saw(1), man(2). Thus, the feature vector for  $(t_i)$  would be  $\vec{v}(t_i) = (2, 1, 2)$ . A VSM consists of one or more feature vectors that occur within a document collection (of e.g., a domain). The aim of text represented in numeric vectors is to deal with them with linear algebraic operations which perform significantly faster (Aggarwal and Zhai, 2012a) than text operations (e.g., text similarity).

### 2.6.3 Knowledge Discovery in Text

Piatetski and Frawley (1991) consider knowledge discovery as the "[...] *nontrivial extraction of implicit, previously unknown, and potentially useful information from data*". Based on the transformed text, existing machine learning and data mining techniques can be applied in order to uncover desired knowledge.

**Machine learning** (ML), in general, refers to processes of improving a machines' performance on a specific task without being explicitly programmed. In detail, an algorithm learns from existing data and is able to make predictions on (new) unknown data (Bishop, 2006). ML basically provides supervised and unsupervised learning methods. Supervised learning methods rely on training data sets. **Training data** consists of already labeled "learning" data. Each record in the training data set is a tuple which consists of input and corresponding expected output (label). The main drawback of supervised ML algorithms is the availability of training data or the huge effort for its

preparation, respectively. The construction of a training data set is time consuming on the one hand and requires domain-specific knowledge on the other hand. Based on the training data, a supervised learning algorithm "learns" to predict the output of unseen new data (see, e.g., Aggarwal and Zhai, 2012a; Manning et al., 2008). For example, many PoS parsers (see Section 2.5.3) are based on supervised learning algorithms (Jurafsky and Martin, 2014). Most text classification algorithms make use of supervised learning methods. Thus, the problem of supervised learning in context of text data is often also referred to as classification (Aggarwal and Zhai, 2012a). On the other hand, unsupervised learning methods do not require any training data. Thus, they can be applied without any manual effort. The two main unsupervised learning methods which are commonly used in the context of text mining are clustering and topic modeling (see, e.g., Allahyari et al., 2017). This thesis also uses information extraction approaches (which partly use ML technologies) in order to discover specific knowledge from text. The next sections introduce the approaches and technologies used in the thesis, namely information extraction, text classification, and text clustering.

### 2.6.3.1 Information Extraction

Information Extraction (IE) aims to automatically process unstructured data in order to identify information (see, e.g., Bender, 2013; Boonthum-Denecke, 2011; Wimalasuriya and Dou, 2010). Thus, IE transforms unstructured data (embedded in text) into structured information (see, e.g., Aggarwal and Zhai, 2012a; Jurafsky and Martin, 2014). *Unstructured data* does not imply structural incoherence, but rather encoded information which makes it difficult for machines to interpret it (Boonthum-Denecke, 2011). IE relies on extracting entities and the relations between those entities, where both the relations and the entities are expressed by words (Bender, 2013). In context of the thesis, only entities (i.e. domain terms) but no relationships between them are extracted (see Section 4.3.4). In general, four categories of domain term extraction methods can be distinguished (Venu et al., 2016). Statistical methods compute the importance of a term within a domain by means of statistical measures (e.g., term frequency, inverse document frequency). These methods might fail to capture the importance of infrequent domain terms. Linguistic methods identify domain terms based on syntactic patterns (e.g., noun + noun, adjective + noun) or lexico-syntactic patterns (e.g., "including" + noun). In general, the definition of these patterns is usually time consuming and tedious. Term extraction approaches are usually supervised ML-based approaches (Kim et al., 2009). Some important and commonly used approaches make use of Naive Bayes and Support Vector Machine classifiers (see Section 2.6.3.2). Graph-based methods in context of information extraction aim to compute the importance of terms within a domain by

means of the quality and quantity of their relationships (e.g., co-occurrence, semantic, syntactic) to other terms. Important terms are considered to be domain-specific.

### 2.6.3.2 Text Classification

Given a set of classes, text classification aims to determine which class(es) a given piece of text belongs to (Manning et al., 2008). Hard text classification assigns exactly one class label to a given piece of text. On the other hand, soft text classification computes a probability distribution for a given piece of text over all the classes. In the following, the two supervised classification algorithms which are used in context of SOFEX, named Naive Bayes and Support-vector machines, are described.

The **Naive Bayes** (NB) classifier is a robust (Manning et al., 2008) supervised text classifier which belongs to the group of probabilistic classifiers and is based on Bayes' theorem (Langley et al., 1992). Probabilistic classifiers gained a lot of popularity recently due to their remarkable performance (Allahyari et al., 2017). The NB classifier models the distribution of PoT in each class using a probabilistic model with independent assumptions about the distribution of words (Aggarwal and Zhai, 2012a). In context of the thesis, a multinomial model for NB is used which captures the frequency of terms in a PoT as BoW. In contrast, a Multivariate Bernoulli Model ignores the frequency of words for classification (Allahyari et al., 2017).

**Support-vector machines** (SVM) belong to the group of supervised linear classifiers. The aim of a SVM is to find an optimal decision boundary (i.e. hyperplane) between two classes that is maximally far from any point in the training data (see, e.g., Manning et al., 2008). SVM then classifies new data by deciding to which side of the defined hyperplane the data belongs to.

### 2.6.3.3 Text Clustering

Text clustering aims at segmenting a set of documents into groups or cluster where documents in the same cluster are similar to each other and dissimilar to documents in other clusters (see, e.g., Hotho et al., 2005). The similarity between documents is measured by means of similarity functions. Traditional methods for clustering originate from the data mining field and are thus usually focused on quantitative data with numeric attributes (see, e.g., Aggarwal and Zhai, 2012a; Allahyari et al., 2017). Thus, vector-based document representations can be used to apply these methods. According to Allahyari et al. (2017), text clustering algorithms can be divided into several different categories. In the following, only the clustering algorithms which are used in context of this thesis are described.

**k-means** (KM) clustering is a widely used distance-based partitioning algorithm for efficient clustering (Aggarwal and Zhai, 2012a). A previously known number of  $k$  groups is formed from a set of similar objects. Initially, it uses  $k$  representatives (seeds) around which the clusters are built. Iteratively, new objects are assigned to those clusters so that the mean across clusters are as different from each other cluster as possible. Furthermore, the seeds (i.e. centroids) are optimized. The "distance" from a centroid to other objects is usually calculated by means of the euclidean distance (straight line between two points in a  $n$ -dimensional space).

**Expectation-Maximization** (EM) clustering is similar to KM clustering and belongs to the probabilistic clustering algorithms. In contrast to KM which aims to maximize the differences of cluster means, the EM algorithm computes probabilities of cluster memberships based on one or more probability distributions in the expectation step. Given the clusters, the goal of the EM algorithms is to maximize the overall probability (i.e. likelihood) of the data in the subsequent maximization step. The expectation and maximization steps are performed iteratively (Celeux and Govaert, 1995). Both, KM as well as EM assume that the number of clusters is previously known.

## 2.7 Measurement Fundamentals

In context of the thesis, five process steps of SOFEX are evaluated with respect to a corresponding ground truth which is called gold standard. The gold standards were manually created by domain experts based on the GDC user manual. In each evaluation, the outcome of a component is compared to its corresponding gold standard. In this thesis two different types of measures which are used within evaluation are distinguished, namely binary and relevance measures. In context of text classification, the aim of the evaluation is to determine if the algorithm correctly identifies a piece of information (e.g., sentence) to correspond to a class or not (e.g., if the sentence contains software feature-relevant information or not). Such classification can either be correct or not. Thus, classification results end up in four different relevance measures:

1. True Positives ( $TP_i$ ) are documents which are classified as and actually are of class  $i$ .
2. False Positives ( $FP_i$ ) are documents which are classified as but actually are not of class  $i$ .
3. False Negatives ( $FN_i$ ) are documents which are classified as type  $j \neq i$  but actually are of class  $i$ .
4. True Negatives ( $TN_i$ ) are documents which are classified as and actually are of class  $j \neq i$ .

Based on these four relevance measures, the recall, precision, and the  $F_\beta$ -score can be calculated. **Precision** measures how many pieces of information are found correctly. In contrast, **recall** measures how many relevant pieces of information are found. The  **$F_\beta$ -score** considers both the precision and the recall as their harmonic mean. Summarized, the recall, precision, and the  $F_\beta$ -score are calculated as follows:

$$P_i = \frac{TP_i}{TP_i + FP_i} \quad R_i = \frac{TP_i}{TP_i + FN_i} \quad F_{\beta_i} = (1 + \beta_i^2) * \frac{P_i \times R_i}{(\beta_i^2 \times P_i) + R_i}$$

The subordinated  $i$  in precision, recall, and  $F_\beta$  refers to the actual type to be investigated in the corresponding evaluation context: in context of terminology extraction, information identification, and information extraction,  $i$  refers to domain terms, software feature-relevant sentences, and atomic software feature-relevant information respectively.

Besides the relevance measures for text classification tasks, evaluations based on binary measures seek to determine if an outcome corresponds to the gold standards. Thus, the result can either be true or false (e.g., a parse tree as the outcome of a PoS parser is either correct or incorrect).

## Part II Problem Investigation

# Chapter 3

## Software Feature Extraction from Natural Language Text: State of the Art

This chapter provides a systematic literature review (SLR) on the extraction of feature-relevant information from natural language software engineering artifacts. It contains an overview of the the SLR process, describes the analytic dimensions to categorize relevant literature, as well as a concluding evaluation of the selected publications. The primary focus of the SLR is to overview published approaches in the context of feature-relevant information extraction from different natural language software engineering artifacts and thus to determine the current state of practice.

Section 3.1 describes the reasoning behind the SLR, the motivation for our research questions, as well as their derivation and inter-dependencies. Then, Section 3.2 introduces the research strategy and process, while Section 3.3 provides an analysis of the selected publications based on characteristics of these approaches. Finally, 3.4 summarizes and concludes the SLR.

### 3.1 Research Questions

Petticrew and Roberts (2006) and Kitchenham and Charters (2007) suggest to use the five PICOC criteria to frame research questions, specify primary objectives and the general scope. The PICOC criteria as well as their values in the context of this SLR are summarized in Table 3.1. PICOC stands for Population, Intervention, Comparison, Outcome, and Context. *Population* refers to the target group of the investigation. *Intervention* specifies the investigation aspects or the issues of interest to the researcher. *Comparison* aims at describing the aspect of the investigation the intervention is being compared to. *Outcome* determines the effect of the intervention and thus the key investigation results of the

SLR). Finally, *Context* refers to the setting or environment of the investigation.

Table 3.1: PICOC criteria

PICOC criteria	Criteria values
Population	software engineering
Intervention	feature extraction from textual sources
Comparison	none
Outcome	extracted entities, mined source, manual effort, constraints, realizability and technologies
Context	empirical papers in industry and academic environment

The research questions (see Table 3.2) are formulated based on the PICOC criteria. In context of this SLR, Population and Intervention are used to frame the main research question RQ#1 (see Table 3.2). The Outcome is used to investigate and formulate detailed research questions (see RQ#1.1 to RQ#1.5 in Table 3.2) in order to gain specific insights into the approaches found in the SLR. Finally, the Context is used to determine the inclusion and exclusion criteria for this SLR (see Table 3.4 in Section 3.2).

Table 3.2: Research Questions for the SLR

RQ	RQ Details
RQ#1	Which approaches exist to extract software feature-relevant information from natural language software engineering artifacts?
RQ#1.1	Which feature-relevant entities are extracted?
RQ#1.2	Which types of natural-language software engineering artifacts are mined?
RQ#1.3	Which technologies are used?
RQ#1.4	Which degree of automation do the approaches provide?
RQ#1.5	Which supporting manual effort is needed to apply the approaches?

The main research question RQ#1 investigates different approaches to extract software feature-relevant information from natural-language software engineering artifacts. In detail, its purpose is to uncover and characterize feature-relevant information (FRI, see RQ#1.1) being extracted (e.g., features, requirements), software engineering artifacts (SEA, see RQ#1.2) which are mined (e.g., product description, user manual), technologies (TEC, see RQ#1.3) used in the approaches (e.g., clustering, classification), the degree of automation (AUT, see RQ#1.4) an approach provides (manual, semi-automated or automated), as well as required manual effort (MAN, see RQ#1.5) in order to apply an approach (e.g., training set, domain ontology). The interdependencies between the characteristics of feature-relevant information extraction approaches is depicted in Figure 3.1.

An information extraction approach itself necessarily mines a textual software engineer-



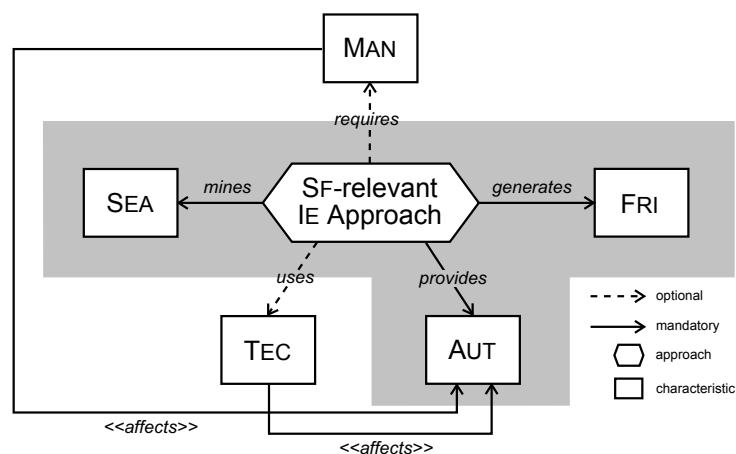


Figure 3.1: Software Feature-relevant Information Extraction Characteristics

ing artifact (SEA) and generates software feature-relevant information (FRI). Furthermore, the process provides a specific degree of automation (AUT). These characteristics are common building blocks of each feature-relevant information extraction approach and are thus mandatory (grey shade in Figure 3.1). The other characteristics (TEC, MAN) are optional because an approach neither necessarily requires a technology in case it is fully manual or manual effort in case it is fully automated. In case the information extraction process uses one or more NLP-specific technologies (TEC), the approach’s degree of automation is either semi- or fully automated. Otherwise, the the approach’s degree of automation is manual per definition. Some information extraction processes require additional manual effort (MAN) to work properly (e.g., training set for text classification, input restrictions). Therefore, both TEC and MAN affect AUT.

## 3.2 Review Method

Kitchenham and Charters (2007) define a systematic literature review as the process of identification, assessment, and interpretation of all relevant information aiming at answering a specific research question. A SLR allows to synthesize information by specifically using inclusion and exclusion criteria to limit and focus the scope of the review in a systematic way. In the following, the overall literature search process is explained.

Kitchenham and Charters (2007) propose using online databases (1) to initially identify relevant literature. Several constraints (e.g., used terms, binary operators, meta data, stop words, etc.) in the interfaces of the different online databases prevent using a standardized, yet detailed search string. Hence, Kitchenham and Charters (2007) suggest a complementary citation-based search (2). Additionally, Jorgensen and Shepperd (2007)

recommend to incorporate a manual target search (3) on key journals and conferences in order to double check and thus minimize the risk of missing relevant literature.

In order to search online databases, a representative search string is required. The search string is determined based on the keywords derived from the values of PICOC's Population and Intervention. These terms are expanded by synonymous and alternative terms (see Table 3.3(a)).

Table 3.3: Definition of search terms

(a) Key words from PICOC criteria with alternatives and synonyms

Term	Alternatives/Synonyms	Set	Connection
feature	capability, function, requirement	= F	} NEAR/5
extract	mine, identify, detect, discover	= E	
text	natural language	= L	} AND
software	system	= S	

(b) Abstract Search String

```
((feature* OR function* OR capabilit* OR requirement*) NEAR/5 (extract* OR min* OR identif* OR detect* OR discover*)) AND (text* OR "natural language") AND (software OR system*)
```

(c) Detailed Search String from function\* NEAR/5 extract\*

```
((function NEAR/5 extract) OR (functionality NEAR/5 extract) OR (functions NEAR/5 extract) OR (function NEAR/5 extraction) OR (functionality NEAR/5 extraction) OR (functions NEAR/5 extraction) OR (function NEAR/5 extracting) OR (functionality NEAR/5 extracting) OR (functions NEAR/5 extracting)) AND (text* OR "natural language") AND (software OR system*)
```

The alternative and synonymous terms are combined by the boolean OR (e.g., `text OR natural language`) resulting in different sets (e.g., L in Table 3.3(a)). In order to limit the number of retrieved literature, the sets F and E are combined via the NEAR operator (= F#E) in order to ensure their narrow relationship in the SLR. The expression `feature NEAR/5 extract` finds articles with “*feature*” within 5 words of “*extract*”; “*extract*” can appear before or after “*feature*”. The sets F#E, L, and S are finally connected via the boolean AND leading to the search string depicted in Table 3.3(b). That search string shows an abstract version containing wildcards (\*) to find related terms (e.g., *funcionalit\** matches *functionality* and *functionalities*). For the specific online databases, the abstract search string is then partitioned into several specific search terms because of database-related notation restrictions (e.g., no wildcard and other boolean operators with NEAR operator). Thus, 20 separate detailed search strings are derived from the abstract search string (see Table 3.3(c)).

In order to filter and determine relevant publications, several inclusion as well as exclusion criteria are defined (see Table 3.4). A publication needs to fulfill the inclusion criteria to be considered relevant. In contrast, a publication is excluded, when it fulfills an exclusion criteria. The exclusion criteria E#1 aims to filter articles which extract classes, relation between entities (e.g. relation between classes), variabilities and commonalities in context of software product lines, aspects or use cases because these entities are not considered to be software feature-relevant. Additionally, exclusion criteria E#2 aims to filter articles which extract software feature-relevant information from structured natural language requirements specifications (RS) only. A structured requirements specification in that context is defined to contain at least one sentence (see, e.g., Boutkova and Houdek, 2011) where each sentence describes a single requirement. Hence, corresponding approaches do not require to filter software feature-irrelevant sentences before extracting software feature-relevant information from software feature-relevant sentences. Thus, these approaches are out of scope of this SLR.

Table 3.4: Inclusion (I#) and Exclusion Criteria (E#)

Criteria #	Description
I#1	Articles describing an approach to extract software feature-relevant information from unstructured natural language software engineering artifacts
E#1	Articles related to class, relation, variability, commonality, aspect, use case extraction
E#2	Articles related to the extraction of feature-relevant information from structured natural language requirements specification documents only
E#3	Proposals, not peer-reviewed, posters, lecture notes, summary of conference keynotes, work in progress, doctoral symposium papers, short papers: concepts which are described in these publications are usually not empirically validated
E#4	Papers not written in English or German
E#5	Papers published before 2000

Finally, all remaining publications are screened based on title and abstract. In the case neither title nor abstract are sufficient to determine the relevancy of a specific publication, the full text is considered. If a publication fulfills I#1 and is not filtered due to fulfilling an exclusion criteria, it is considered for the SLR.

Based on the findings of the online database search, a citation-based search is conducted by means of Google Scholar. The backward search takes all publications which are referred by the SLR publications into account, while the forward search considers the publications which reference the SLR publications. As a third part of the literature search, a manual target search was performed, as it might yield high-quality search results (Jorgensen and Shepperd, 2007) and helps to prevent overlooking relevant literature (in the sense of a verification). Therefore, the most relevant journals and conferences in requirements and

Table 3.5: Journals and Conferences for manual target search (alphabetically ordered)

Journals	ACM Computing Surveys, ACM Transactions on Software Engineering and Methodology, Automated Software Engineering Journal, IEEE Software, Empirical Software Engineering Journal, IEEE System Journal, IEEE Transactions on Software Engineering, Information and Software Technology, Journal of Systems and Software, Requirements Engineering Journal
Conferences	International Conference on Software Engineering (ICSE), International Symposium on Foundations of Software Engineering (FSE), International Working conference on Requirements Engineering for Software Quality (REFSQ), Requirements Engineering Conference (RE), International Conference on Mining Software Repositories (MSR), International Conference on Software Analysis, Evolution, and Reengineering (SANER)

software engineering are investigated. In total, 11 journals as well as 6 conferences are manually explored (see Table 3.5). All papers published between 2000 until November 2017 in the selected venues were considered.

### 3.3 Results

This section presents the results of the SLR: first, the application of the search strategy is described, followed by a categorization of the selected publications to answer the research question RQ#1 from Table 3.2.

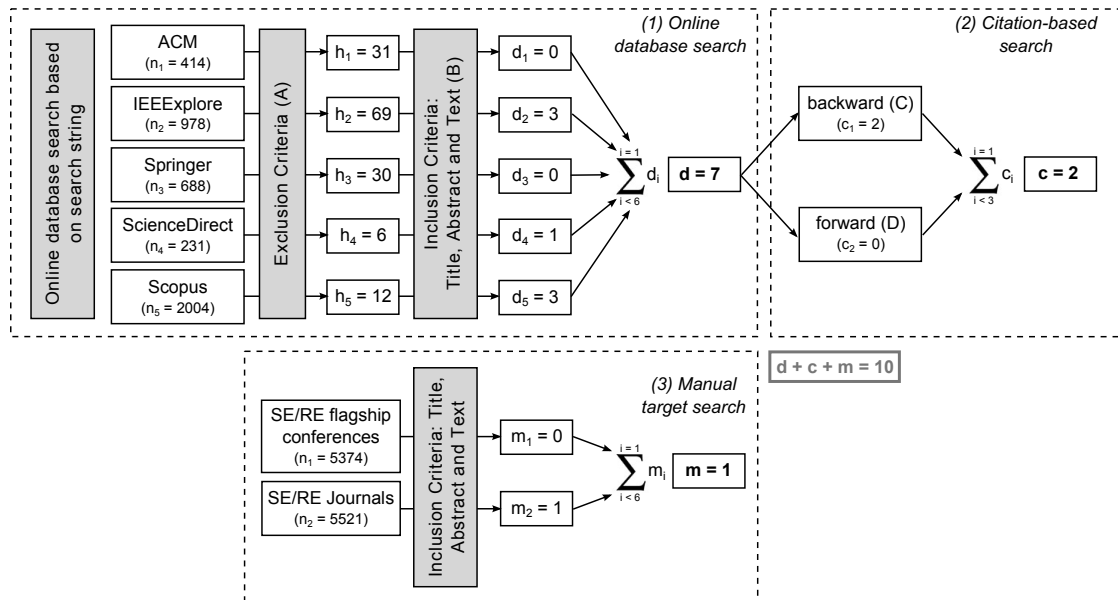


Figure 3.2: Process of the Literature Review

Initially, using the detailed search strings, five different online databases which are relevant for software engineering were queried: IEEE Explore, ACM Digital Library,

Springer, ScienceDirect and Scopus. The visualization of the application of this search strategy with the results of the search is presented in Figure 3.2.

The online database search (1) returned 4315 hits in total (including duplicates). In a first step, the exclusion criteria (see Table 3.4) were applied (A) which resulted in 148 remaining publications. After verifying the inclusion criteria by screening the titles, abstract, and text (B), 7 publications were considered relevant. Based on the publications from the online database search, a citation-based search was conducted (2) on 217 unique references. Backward (C) and forward snowballing (D) yielded 2 additional publications in total. In course of the manual target search (3) more than 5500 titles were screened in journals and about 5400 articles in the selected conferences and corresponding workshops. However, after removing duplicates there was only one additional study found. This indicates that our search string for the online database search is reliable and the citation-based search was sufficient.

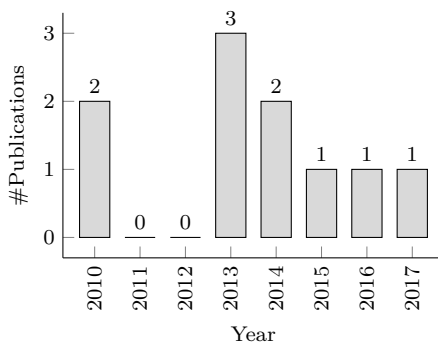


Figure 3.3: Publication Chronology

The selected publications originate from 9 different publication venues with the Requirements Engineering Conference (RE) as the most popular venue (2 publications). Duplicates were found for Hariri et al. (2013), an older publication at ICSE in 2011 (Dumitru et al., 2011) and a publication from 2013 in IEEE Transactions on Software Engineering. In order to avoid duplicates, the more comprehensive version from 2013 was considered. Finally, a total of 10 publications were selected to be con-

sidered in the SLR (see RQ#1 in Table 3.2). Figure 3.3 illustrates the distribution of the selected publications from 2010 to 2017 with 2013 as the main contributor.

### 3.3.1 Findings in Selected Publications

This section describes the examination of the selected publications for this SLR and answers the research question along the key characteristics of the corresponding approaches. Table 3.6 shows an overview as well as a unique ID for the publications which were selected in course of the systematic literature review. The IDs are used to help distinguishing between the publications selected for the SLR and the references in this thesis, and to increase the readability of the tables.

In general, the relevant publications are investigated regarding the characteristics of the presented approaches to extract software feature-relevant information. These characteristics as well as their possible values (*value set*) are listed in Table 3.7. The

Table 3.6: Publication Venues of the selected Studies

ID	Reference	ID	Reference
P1	Bakar et al. (2016)	P6	Khan et al. (2014)
P2	Guzman and Maalej (2014)	P7	Li et al. (2015)
P3	Hariri et al. (2013)	P8	Slankas and Williams (2013)
P4	Johann et al. (2017)	P9	Yu et al. (2013)
P5	John (2010)	P10	Zhan and Li (2010)

characteristics in the table correspond to the characteristics depicted in Figure 3.1. The value sets of the corresponding characteristic are derived from the articles of the SLR.

Table 3.7: Characteristics of the SLR Publications

Charact.	Description	Section
EXT (RQ#1)	Extracted software feature-relevant entities <i>Value set</i> = {Functional Requirement (FR), Non Functional Requirement (NFR), Feature (F)}	3.3.1.2
SEA (RQ#1)	Type of software engineering artifact, which is used as source for information extraction <i>Value set</i> = {Requirement Specification (RS), Manual/Product Description (MD), Review (R), Other Text Document (D)}	3.3.1.2
TEC (RQ#1.1)	NLP characteristics <i>Value set</i> = {Preprocessing (PP), Information Identification + Extraction (IX), Text Clustering (CG), Text Classification (CL), NLP Tool (TO)}	3.3.1.3
AUT (RQ#1.2)	Automation of the approach <i>Value set</i> = {manual (M), semi-automated (S), automated (A)}	3.3.1.4
MAN (RQ#1.3)	Manual preparation required to enable the realization of the approach <i>Value set</i> = {Domain Analysis (DA), Domain Ontology (DO), Human Partizipation (PA), Text Correction (TC), Training Set (TR), Input Restriction (IR)}	3.3.1.5

### 3.3.1.1 Which approaches exist to extract software feature-relevant information from natural language software engineering artifacts?

This section investigates existing approaches to extract software feature-relevant information from natural language software engineering artifacts and aims to answer RQ#1. In the following, this section describes each approach, found in the SLR in detail.

P1 (Bakar et al., 2016) describes an approach to extract features from online software reviews. In a first step, they identify similar documents by means of the Fuzzy C-Means (FCM) clustering algorithm where each data point has a probability of belonging to each cluster. A data point which is located closer to a clusters' centroid has a stronger membership to the cluster (Bezdek et al., 1984). In a second step, they extract bigrams

and trigrams (which represent features) based on PoS patterns from each cluster. Finally, the approach clusters the extracted features into similar feature clusters by means of a modified Word Overlap Metric.

P2 (Guzman and Maalej, 2014) provides an approach that allows to automatically identify application features mentioned in user reviews. They assume that nouns, verbs, and adjectives are most likely used to describe features (in contrast to e.g., adverbs, numbers, or quantifiers). From the words which are tagged as noun, verb, or adjective, the approach extracts features by means of a collocation algorithms. A collocation algorithm in context of text mining identifies words which unusually often co-occur (see, e.g., Aggarwal and Zhai, 2012a). The approach finally considers only collocated words with less than three words and appear in at least three reviews. The authors use Latent Dirichlet Allocation (LDA) to finally group extracted features that tend to co-occur in the same reviews and furthermore assign topics to each review. LDA is a probabilistic distribution topic modeling algorithm which automatically discovers topics a corresponding document contains (Blei et al., 2003).

P3 (Hariri et al., 2013) provides an approach to extract common software features across products from online product listings. Therefore, they use an incremental diffusive clustering (IDC) algorithm that incrementally identifies features based on a voting schema using distance metrics.

The approach from P4 (Johann et al., 2017) allows to extract high-level features (describing essential functional capabilities) from app descriptions and app reviews. They use predefined PoS patterns (e.g., VB NN NN) to extract feature candidates. Furthermore, they simplify sentences which include enumerations and conjunctions in order to provide atomic sentences to be used for feature extraction. The feature candidates which are extracted from an app description and the feature candidates which are extracted from the related user review are used to match these features by means of a binary text similarity function on different levels of granularity. First, the similarity function determines the similarity between the feature candidates based on single word matching. Second, the approach uses WordNet to consider synonym sets of captured words (e.g., photo and image). Finally, to compensate a possible difference in the number of words, the approach utilizes cosine similarity.

P5 (John, 2010) proposes CAVE, a fully manual approach to identify software feature-relevant information in NL user documentation. Therefore, they determine a set of patterns (e.g., section headings typically contain features, repeated words or phrases can be domains or subdomains), which allows to locate software feature-relevant information.

The approach from P6 (Khan et al., 2014) allows to identify product features in customer reviews by means of syntactic patterns. They extract "base noun phrases",

linking "verb based noun phrases", and "preposition based noun phrases" that represent product features by means of different syntactic patterns.

P7 (Li et al., 2015) provides an approach to automatically extract requirements for scientific software from available NL knowledge sources like user manuals and project reports. They use a combination of syntactic (PoS) and lexical patterns (e.g., *method of {NN | NP}*) as well as a gazetteer<sup>1</sup> in order to identify and extract requirement candidates of different DRUMS (Domain specific Requirements Modeling for Scientists) types. The DRUMS types define core requirement types (e.g., data definition, interface, process) in the DRUMS model. Similar to P2, the authors utilize the LDA topic modeling algorithm to group the extracted DRUMS requirement candidates. The LDA algorithm performs the clustering task based on bi- and trigrams which are determined from the requirement candidates by means of a collocation algorithm a priori. Finally, the LDA algorithm computes topics (features) for each requirements cluster.

P8 (Slankas and Williams, 2013) describes an approach to automatically extract non-functional requirements (NFR) from unconstrained NL documentation. The approach utilizes a k-nearest neighbor (*k*-NN) classifier. The *k*-NN classifier is a supervised algorithm which classifies a new object based on which objects previously classified (i.e. training objects) are closest to the new object. The majority class from these *k*-nearest neighbors is defined as class label for the new object (Aggarwal and Zhai, 2012b). The closeness between objects is determined by means of a distance metric (e.g., Euclidean for numerical attributes). The authors use a modified version of the Levenshtein distance (see Levenshtein, 1966). They evaluated the classifier against others (e.g., SMO, NB) and report that SMO performed better than *k*-NN.

P9 (Yu et al., 2013) proposes an approach to mine and recommend software features across multiple software web repositories like *sourceforge.net*. In that context, they created a hierarchical repository of software features (HESA). HESA contains features on high-level which are described by feature elements. A feature element is defined as a "[...] *raw description of a feature which can indicate a functional characteristic or concept of the software product*" (Yu et al., 2013). A feature element corresponds to a sentence of an online software profile. The feature elements are clustered by an extended LDA algorithm into flat clusters. Finally, an improved Agglomerative Hierarchical Clustering algorithm (IAHC) transforms the features (and corresponding feature elements) into a hierarchical (semantic-based) feature structure. An Agglomerative Hierarchical Clustering algorithm is a bottom-up clustering method which determines hierarchical clusters (e.g., clusters have sub-clusters) based on distance metrics in an iterative process.

---

<sup>1</sup>A gazetteer in context of text mining is a domain-specific dictionary used to mine identify domain-specific terms



P10 (Zhan and Li, 2010) provides an approach which mines product features in product reviews by means of their nominal semantic structure. Initially, PoS tags are used in syntactic patterns to determine noun fragments in the product reviews. Starting from the noun fragments and their semantic dependencies, the approach determines potentially relevant non-nominal semantic neighbors that can be either adjectives or verb predicates. The combination of a nominal noun fragment and a non-nominal semantic neighbor represents a product feature. As a last step, the authors apply co-clustering in their approach on product features in order to determine fine-grained product feature cluster.

### 3.3.1.2 Which software feature-relevant entities are extracted? Which types of natural language software engineering artifacts are mined?

This section investigates the different types of software feature-relevant information which are extracted (see RQ#1.1) as well as the different types of natural language software engineering artifacts mined (see RQ#1.2) with the corresponding results summarized in Table 3.8.

Table 3.8: Types of EXT and FRI

P#	EXT			SEA			
	F	FR	NR	RS	MD	R	D
P1	✓			✓	✓	✓	
P2	✓						✓
P3	✓				✓		
P4	✓				✓	✓	
P5	✓				✓		
P6	✓						✓
P7		✓	✓		✓		✓
P8			✓	✓	✓	✓	✓
P9	✓				✓		
P10	✓						✓

The publications of the SLR present approaches which extract three different types of software feature-relevant entities (see EXT in Table 3.8): 8 of the considered approaches extract features, feature models or feature trees (P1-P6,P9, and P10), 1 approach extracts functional requirements (P7), and 2 approaches extract non-functional requirements (P7 and P8). P7 is the only approach which allows the extraction of more than one type of software feature-relevant information (FR and NFR).

On the other hand, the natural-language software engineering artifacts which are used as sources to extract software feature-relevant information are grouped into four different categories (see SEA in Table 3.8) Most approaches (7) mine manuals or product descriptions (P1,P3-P5, and P7-P9), 6 approaches mine (online) software reviews (P1,P2,P4,P6,P8, and P10), and 2 approaches mine requirement specification (P1 and P8) and other documents (e.g., project reports; P7 and P8) respectively. P8 presents the only approach which is not restricted to a specific software engineering artifact to be mined. It allows to mine any natural language document in order to extract non-functional requirements.

### 3.3.1.3 Which technologies are used?

The approaches considered in the SLR use different technologies which are listed in Table 3.9. These technologies are grouped along the characteristics of the approaches: each at least semi-automated approach identifies or extracts software-feature relevant information to some degree with different techniques and uses text clustering and/or text classification by means of one or more NLP tools. Furthermore, each (semi-)automated approach uses at least one text preprocessing technique.

Table 3.9: NLP Characteristics of the SLR Publications

NLP Classification	Characteristic	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Text Preprocessing	Tokenization	✓	✓	✓	✓		✓	✓	✓	✓	✓
	Stopwords		✓	✓	✓			✓	✓	✓	✓
	PoS Tagging/Parsing	✓	✓		✓		✓	✓	✓		✓
	Stemming			✓							✓
	Lemmatization	✓	✓					✓	✓		
	Dependency Parsing/Role Labeling								✓		✓
	Tf-Idf	✓		✓							
Information Identification & Extraction	Classification								✓		
	Clustering	✓	✓	✓	✓			✓		✓	✓
	Syntactic Patterns	✓	✓		✓		✓	✓			✓
	Lexical Patterns							✓			
	Semantic Patterns										✓
Text Clustering	Agglomerative/Hierarchical										✓
	Distance-based Partitioning	✓		✓	✓						
	Co-Clustering							✓			✓
	Probabilistic		✓					✓		✓	
	Network-based	✓									
Text Class.	Proximity-based				✓				✓		
NLP Tools	Weka <sup>2</sup>								✓		
	NLTK <sup>3</sup>	✓	✓		✓						
	Stanford NLP <sup>4</sup>								✓		
	WordNet <sup>5</sup>				✓			✓	✓		
	Gate <sup>6</sup>							✓			
	OpenNLP <sup>7</sup>										✓
	Mallet <sup>8</sup>									✓	
	Unknown			✓			✓				

<sup>2</sup><https://www.cs.waikato.ac.nz/ml/weka/>

<sup>3</sup><http://www.nltk.org/>

<sup>4</sup><https://stanfordnlp.github.io/CoreNLP/>

<sup>5</sup><https://wordnet.princeton.edu/>

<sup>6</sup><https://gate.ac.uk/>

<sup>7</sup><https://opennlp.apache.org/>

<sup>8</sup><http://mallet.cs.umass.edu/>

As P5 describes a fully manual approach, there is obviously no computer-aided technique involved and will thus not be considered in this section. All the other approaches which are identified in the SLR are machine-aided to a certain degree.

**TEXT PREPROCESSING.** Most approaches use tokenization (9), stopword removal (7), and PoS tagging or parsing (7). Furthermore, 6 approaches use some kind of "word normalization" (stemming and lemmatization) in order to increase the text mining performance. Last, two approaches use Tf-Idf text representation and two other approaches use semantic parsing (dependency parsing or role labeling).

**INFORMATION IDENTIFICATION & EXTRACTION.** In order to identify and extract software feature-relevant information, only two approaches use supervised classification (P3 and P8). Unsupervised text mining mechanisms are used in 8 approaches: 7 approaches use text clustering and 6 approaches use syntactic patterns. In general, all 6 approaches that use syntactic patterns (P1,P2, P4,P6,P7, and P10), identify and extract features and feature-relevant information (e.g., requirements, product features) based on syntactic patterns that typically include nouns combinations. There is only one approach which uses lexical patterns (P7). Another approach uses semantic patterns (P10).

**TEXT CLUSTERING.** Several approaches (7) use some kind of clustering technique. Clustering is used to group similar features or feature-relevant information. Furthermore, three approaches (P2,P7, and P9) use the LDA topic modeling algorithm in order to "describe" the corresponding clusters.

**TEXT CLASSIFICATION.** There is only one approach which uses text classification (P8). The main reason may be that the effort to apply classification appropriately is extremely high compared to other techniques (e.g., clustering, syntactic patterns) due to the usually manual preparation of training data.

**NLP TOOLS.** The range of NLP tools that are used in the approaches vary widely: NLTK (P1,P2, and P4), Weka and Stanford NLP (P8), WordNet (P7 and P8), Gate (P7), OpenNLP (P10), and Mallet (P9). P3 and P6 make no statements about the tools which are used in their approaches. The approach in that thesis uses Stanford NLP for text preprocessing and information extraction as well as Weka for ML-based tasks.

#### **3.3.1.4 Which degree of automation do the approaches provide?**

P5 present the only approach which is applied manually. P1, P7, and P8 provide semi-automated approaches while P2, P3, P4, P6, P9, and P10 provide fully automated approaches.

### 3.3.1.5 Which supporting manual effort is needed to apply the approaches?

The classifier used in the approach from P8 in order to automatically extract relevant non-functional requirements from unconstrained NL documentation requires manually created training data. The approach from P7 in order to automatically extract requirements for scientific software requires to manually define a list of keywords for the gazetteer. Additionally, the approach considers a data input preparation which requires to manually remove textual noise such as table of contents or references.

## 3.4 Summary and Conclusion

The SLR provides detailed insights into existing approaches which aim to identify and extract software features or software feature-relevant information from NL software engineering artifacts. Based on the insights gained, it is now possible to design an approach which allows to meet the goals defined in Section 1.3 as well as the requirements which are to be described in Section 4.1.

In general, each (semi-)automated approach uses some kind of preprocessing in order to provide a normalized text data basis for further analysis: tokenization, stopword removal, PoS tagging and parsing, as well as stemming or lemmatization are superficially used.

In contrast to SoFEX, most approaches from the SLR do not require the separation of relevant from irrelevant text before starting the analysis because they either consider the entire text to be relevant (P2, P3, P4, P9) or the approach is independent of relevant sentences (P1, P6, and P10) at all. There are only two approaches which provide support for the identification of relevant text data (P7 and P8).

In order to determine relevant information, most approaches use text clustering algorithms and syntactic patterns. Basically, these approaches do not require high manual effort in contrast to approaches which use classification and thus require manually created training data set. Lexical patterns are very specific in contrast to syntactic patterns and may tend to overfit an approach (e.g., P7) if specific words (e.g., “method of”) do not appear in a text to be analysed (e.g., different domain, no template used).

## Part III Treatment Design

## Software Feature Extraction (SOFEX)

The aim of this thesis is to develop an approach to identify and extract software feature-relevant information from unconstrained NL user manuals to enrich existing software features knowledge. This chapter details the design of SOFEX which is based on the goals outlined in Section 1.2 and additional requirements expressed by Roche (see Section 4.1). Section 4.2 provides an overview and describes the overall design of SOFEX in accordance to the named requirements. Sections 4.3 to 4.5 describe the different steps of SOFEX and the technologies used in detail. Furthermore, rationales for the usage of specific technologies in relation to the findings of the articles from the SLR are provided.

### 4.1 Requirements

This thesis is based on a cooperation with Roche Diagnostics GmbH. In general, Roche provides a comprehensive and mature, unstructured and unconstrained NL user manual of GDC which is used as an information (and mining) source. Furthermore, Roche provides a hierarchical list of software features with a short NL description of the corresponding software features. Based on these prerequisites, Roche named several requirements that need to be addressed by SOFEX to extract software feature-relevant information from the user manual and enrich existing software feature knowledge with that information. In detail, Roche named the following requirements which refine and extend the goals of SOFEX (see Section 1.2):

R.1 *comprehensive software feature description*: provide a holistic and detailed description of existing GDC software features and corresponding software feature-relevant information in order to get detailed and structured insights in the capabilities of GDC.

R.2 *hierarchical software features*: provide different granularity levels of the GDC

software feature description in order to get insights in more abstract as well as detailed GDC capabilities. Depending on the role, stakeholders are interested in software feature descriptions on different abstraction levels (e.g., a member of the top management might be interested in a more rough grained overview of the system capabilities than a developer).

- R.3 *atomic software feature-relevant information*: the software feature-relevant information should be as simple and short as possible (= atomic) in order to provide simple and unambiguous information at a first glance.
- R.4 *low manual effort*: the approach should be applicable with lowest manual effort in order to apply the approach properly.
- R.5 *high performance*: the approach should provide high recall as well as high precision in order to gather a detailed and as truthful as possible representation of the GDC capabilities.

## 4.2 Overview

In general, gathering and extracting software feature-relevant information from NL user manuals as well as the assignment of that software feature-relevant information to existing software features poses two main challenges: (1) not each sentence of a manual contains software feature-relevant information and (2) the manual's textual content is usually not structured along software features. More precisely, these documents lack relevant domain-specific meta data (Aggarwal and Zhai 2012a). Thus, SOFEX needs to identify sentences which contain software feature-relevant information and extract atomic software feature-relevant information before determining the logically related software feature. SOFEX consists of the three subsequent steps information identification, information extraction, and information assignment (see Figure 4.1). To support and ease these three steps, text preprocessing techniques are applied. As mentioned in Section 2.6.1, text preprocessing aims to remove noise (irrelevant text data) and thus reduce the amount of data to be further processed and is therefore applied in the context of each of the three steps of SOFEX. In general, the user manual is used to identify potentially software feature-relevant sentences (FRS) in a first step. Afterwards, the potentially software feature-relevant sentences are used to extract atomic software feature-relevant information (FRI, see requirement R.3). In a last step, the atomic software feature-relevant information is assigned to corresponding and existing software features (Feature Hierarchy) in order to provide a holistic Hierarchical Feature Description (see requirements R.1 and R.2).

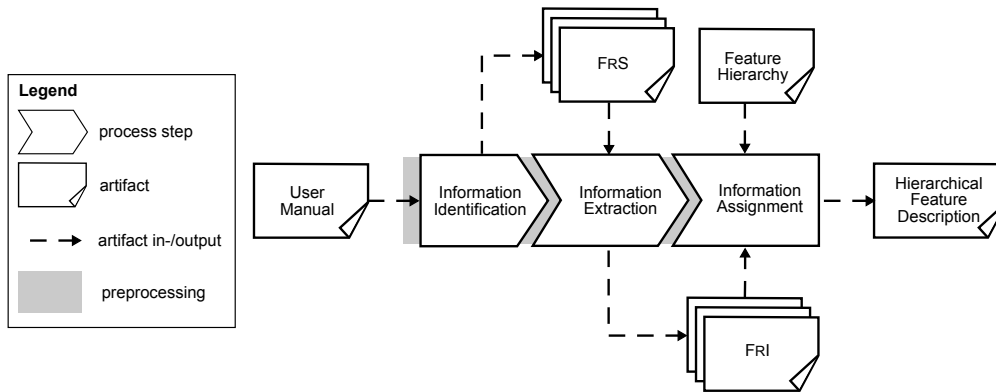


Figure 4.1: SoFeX Process Steps

### 4.3 Information Identification (semi-automated)

In a first step, SOFEX requires to identify pieces of data on sentence-level that convey software feature-relevant information in a NL user manual. In other words, SOFEX needs to separate sentences that potentially contain software feature-relevant information (i.e. potentially software feature-relevant sentences) from sentences which do not contain software feature-relevant information and are thus considered as software feature-irrelevant sentences.

Most articles from the SLR do not provide an approach to determine relevant sentences in text documents because they either only extract phrases which are independent of relevant sentences (P1, P6, and P10) or the entire NL text used for mining is considered to contain only relevant sentences (P2, P3, P4, and P9). Furthermore, P5 is a manual approach and thus does not apply any NLP technology. There are only two approaches which provide support for the identification of software feature-relevant sentences: P7 uses a gazetteer with domain terms which initially allows to indicate sentences which may contain software feature-relevant information. On the other hand, P8 uses a trained classifier to identify sentences which represent non-functional requirements from unconstrained NL documents.

In context of SOFEX, the application of a classification algorithm in order to differentiate software feature-relevant from software feature-irrelevant sentences requires training data. The provision of such training data requires domain experts to label a set of exemplary sentences to be either software feature-relevant or software feature-irrelevant. In order to take requirement R.4 into account, the application of a classification algorithm and thus the provision of training data cannot be considered anymore. Additionally, an analysis of the GDC user manual shows that the identification of software feature-relevant sentences in context of GDC by means of domain-specific terms (gazetteer, see P7) works



with a higher precision and recall (according to R.5) compared to using a classification algorithms in order to determine software feature-relevant sentences; (see Sections 5.2.1 and 5.3.2). Furthermore, SOFEX provides an approach to semi-automatically extract domain-specific terms from NL text documents in order to reduce the amount of manual effort (see Section 4.3.4).

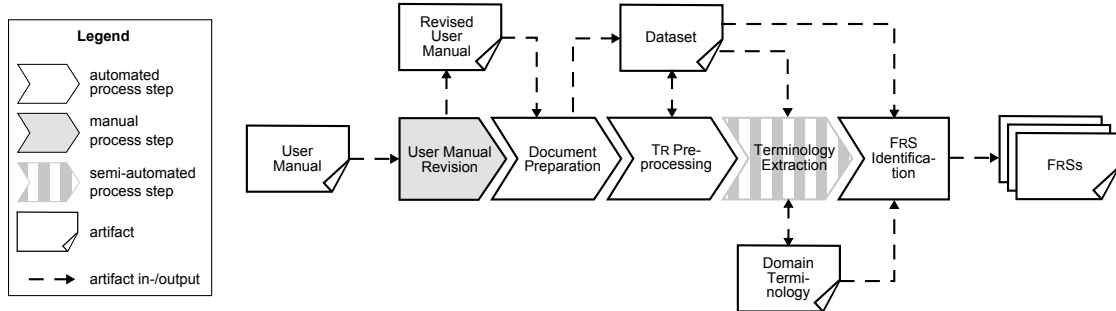


Figure 4.2: Semi-automated Information Identification

The input of information identification is the GDC user manual and the output for further processing is a set of software feature-relevant sentences. Before software feature-relevant sentences can be identified in the user manual by means of NLP, the text needs to be prepared. Similar to all approaches from the SLR which aim to gain knowledge from NL artifacts by means of NLP and ML technologies (see P1-P4 and P6-P10), SOFEX must provide a syntactical correct and cleansed textual basis first to ensure high quality results. Thus, the preparation of the user manual consists of the three main steps user manual revision, document preparation, and terminology-related (TR) preprocessing, which are explained in detail in the upcoming sections.

#### 4.3.1 User Manual Revision (manual, optional)

NLP techniques, especially POS-related techniques, require a syntactically correct textual basis in order to deliver correct and expected results. Therefore, before applying the NLP-related tasks of SOFEX, an initial user manual revision may be necessary in order to ensure syntactical correctness. This process step is optional as it depends on the provided user manual syntax: in case the user manual is already syntactically correct, it is not necessary to revise it. The effort for the revision obviously depends on the quality of the user manual and furthermore the linguistic abilities of the individual to perform that task. Besides information assignment (see Section 4.5.2), this process step is the only fully manual task to be carried out in context of SOFEX. Nevertheless, the revision can be performed by a non domain expert: a student from Translation Studies revised the second section of the GDC user manual.

### 4.3.2 Document Preparation (automated)

The preparation of the document aims to extract the NL test and corresponding structural (meta) information from a user manual written in Microsoft Word (\*.docx).

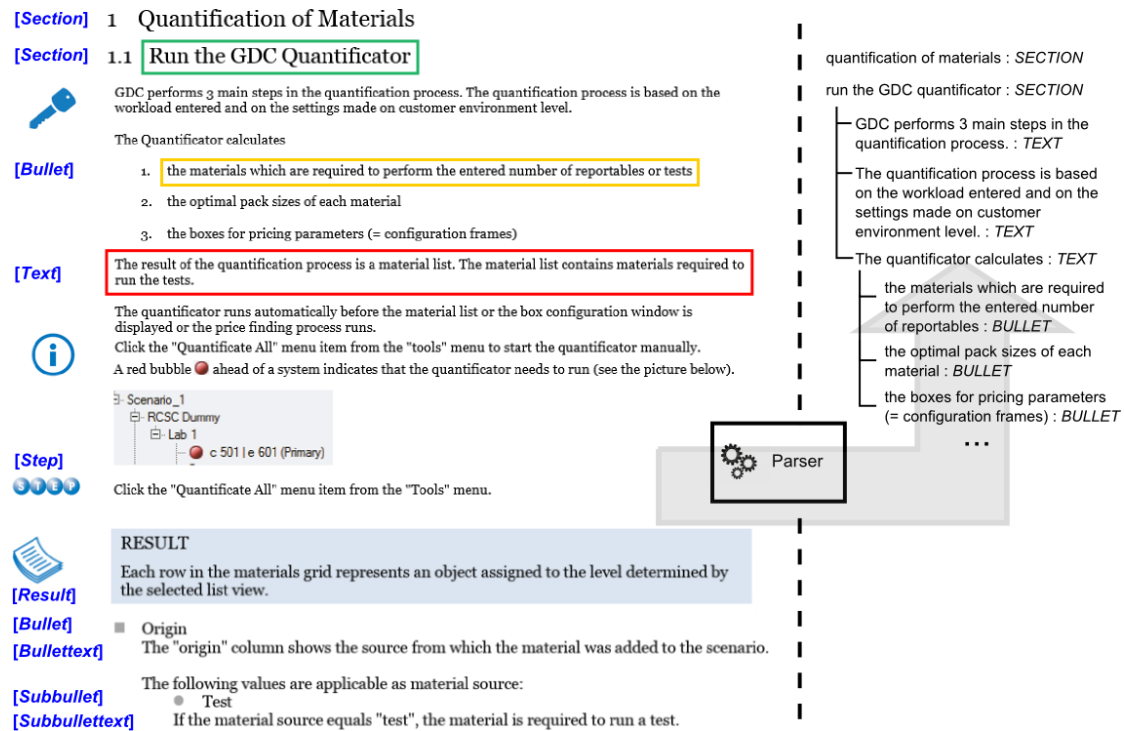


Figure 4.3: Document Preparation (left: user manual excerpt, right: *Dataset*)

The extraction of NL text data and structural information (e.g., bullet point, heading, etc.) from \*.docx files is - in comparison to other document formats (e.g., \*.pdf) - of low complexity by means of open source APIs (e.g., Apache POI<sup>1</sup>). SoFEX extracts and furthermore transforms the NL text into an internal data model (*Dataset*) which eases further SoFEX tasks (see right-hand side of Figure 4.3).

A user manual can contain both NL text and associated images (see left-hand side of Figure 4.3). SoFEX processes NL text only. Basically, NL text is organized in paragraphs which contain at least one sentence (see, e.g., red, green, and yellow rim). A sentence can be either complete (red rim) or incomplete (yellow rim). In contrast to a complete sentence, an incomplete sentence does not necessarily contain a subject or a predicate. Furthermore, a complete sentence conveys a complete thought and thus, does not lack grammatical components (Ward and Woods, 2013).

The content of a user manual is usually structured (e.g., (sub-)sections, bullets, listings) which conveys additional implicit semantic information to the reader. As an example, a

<sup>1</sup><https://poi.apache.org/>

listing may indicate a conjunction of the listed sentences. A structural information of a sentence corresponds to the style sheet of the sentence in the Microsoft Word document. In context of SOFEX the structural information is called **sentence type**. Investigations of the GDC user manual showed that sentence types are not sufficient on their own to identify software feature-relevant sentences like suggested in P5. Nevertheless, SOFEX uses sentence types to identify software feature-irrelevant sentences (see Section 4.3). In context of SOFEX and the GDC user manual, 8 different sentence types were identified (see blue square brackets on the left-hand side of Figure 4.3):

- **text** refers to normal text which is part of a section which does not semantically belong to another sentence type (e.g., bullet, sub-bullet, step, result).
- **bullet** refers to the text after a bullet (bullet heading)
- **bullet text** refers to text below a bullet which semantically belongs to the bullet
- **sub-bullet** refers to the text after a sub-bullet (sub-bullet heading)
- **sub-bullet text** refers to text below a sub-bullet which semantically belongs to the sub-bullet
- **step** refers to the text which belongs to the “step” area in the GDC user manual (indicated by the “step” icon)
- **section** refers to the text which names a corresponding section
- **result** refers to the text which belongs to the “result” area in the GDC user manual (indicated by the “result” icon).

Figure 4.4 shows two examples of style sheets which are used in the GDC user manual: “1Section\_1” refers to the sentence type *section* whereas “GDC Text normal bulletpoint” refers to the sentence type *bullet point*. As mentioned, the sentence types are used in the subsequent process steps (see Section 4.3) in order to reduce the amount of data to be further analyzed (software feature-irrelevant sentences).



Figure 4.4: Sentence Type Examples

SOFEX extracts the NL text as well as the corresponding sentence types from the

user manual by means of the Apache POI API (indicated by *Parser* in Figure 4.3). Furthermore, SOFEX applies some minor automated adjustments to correct wrong sentence types which are wrongly defined by the author of the user manual. The patterns of the wrong sentence types were uncovered while analysing the GDC user manual:

- a sentence of type “text” cannot directly be followed by a sentence of type “sub-bullet” and is therefore changed to type “bullet”
- a sentence of type “bullet” cannot directly be followed by a sentence of type “sub-bullet text” and is therefore changed to type “bullet text”
- a sentence of type “section ” cannot directly be followed by a sentence of type “bullet text” and is therefore changed to type “text”

### 4.3.3 Terminology-related (TR) Preprocessing (automated)

Based on the extracted NL text (sentences) and sentence types in the Dataset, TR preprocessing aims to generate a parse tree for each sentence. But, in order to ensure parse trees of high quality, the NL text needs to be cleansed in order to remove possible textual noise. Thus, SOFEX applies some lexical text modifications:

1. lowercasing (e.g., *GDC* → *gdc*)
2. removing line breaks in sentences
3. indication of quoted phrases.

Lowercasing and removing line breaks are typical preprocessing tasks (see, e.g., Aggarwal and Zhai, 2012a). On the other hand, the indication of complex phrases as a single word (string encoding, see, e.g., Ghosh et al., 2014) is not frequently used in preprocessing tasks. Nevertheless, it allows to increase the accuracy of parse trees generated by means of a POS parser. Quoted phrases are indicated by bundling the words in between the quotes with delimiters and through adding a prefix. Both delimiter and prefix are defined to be “QD”. As an example, the sentence *press the button “test the efficiency”* becomes *press the button QDtestQDtheQDefficiency*. After these textual modifications, the sentences are tokenized and parsed by means of the toolkit Stanford CoreNLP<sup>2</sup>. The Stanford POS parser generates a parse tree for each sentence. The parse trees are then added to the sentence in the *Dataset*.

### 4.3.4 Terminology Extraction (semi-automated)

Terminology Extraction deals with the extraction of a domain-specific terminology (domain terminology in short) from the user manual. A domain terminology consists

---

<sup>2</sup><http://nlp.stanford.edu/software/lex-parser.shtml>

of a set of domain-specific terms (domain terms, see Section 2.4). The domain terms are required to further identify potentially software feature-relevant sentences (see, e.g., Wimalasuriya and Dou 2010) in the GDC user manual similar to P7. Investigations of the GDC user manual showed that the identification of potentially software feature-relevant sentences by means of a domain terms performs with high precision and recall and outperforms other automated approaches (see Sections 5.2.1 and 5.3.2).

Most articles from the SLR provide approaches to extract domain-specific information in general (P1,P2,P4,P6,P9, and P10). All these approaches can be applied fully automated. P1 extracts terms (feature candidates) by means of statistical computation based on Tf-Idf. P2 use graph-based methods to extract terms that correspond to features. The approach identifies important terms by means of a collocation algorithm. P4, P6, and P10 use syntactic and linguistic patterns to identify domain-specific terms that correspond to features. The main problem is that they lack precision, recall or both (see Section 5.3.1). Therefore, SoFEX provides a semi-automatic syntactic pattern-based (see P4, P6, and P10) approach to extract domain terms with high precision and recall (see requirement R.5). Indeed, the effort to extract domain terms increases in comparison to a fully automated approach (see requirement R.4) but the quality of the extracted domain terms significantly outperforms automated approaches which is considered more important.

In a nutshell, Terminology Extraction of SoFEX automatically extracts and presents a list with candidate domain terms to a domain expert who then semi-automatically validates the list meaning who the chooses the true domain terms. An overview of this process is depicted in Figure 4.5. The two process steps within Terminology Extraction, namely Candidate Extraction and Term Validation are described in detail in the following sections.

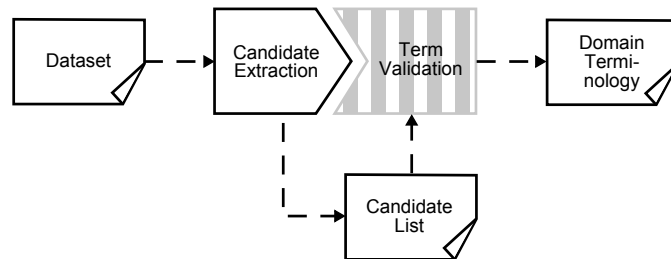


Figure 4.5: Terminology Extraction Process

#### 4.3.4.1 Candidate Extraction (automated)

Candidate Extraction aims to extract domain term candidates (candidates in the following) from the user manual. Therefore, the parse trees of the sentences in the *Dataset* serve as data source. First, by means of POS patterns, all noun phrases (NP, see Table A.1 in

Section A.1 of the Appendix) which contain at least one noun (**NN**, **NNS**, **NNP**, or **NNPS**) are extracted (Tregex pattern **NP < /NN.\*/**) and uniquely stored in a candidate list. In a second step, the list is automatically revised:

1. all words and phrases from the noun phrases whose PoS tag is not equal to either a noun (**NN.?**), a verb (**VB.?**), an adjective (**JJ**), or an adverbial phrase (**ADVP**) are removed
2. all *quoted* phrases which are indicated by the prefix **QD** are removed (see 4.3.3)
3. each single word of each n-gram is added to the candidate list, e.g., based on the word *material list* which is contained in the candidate list, the words *material* and *list* are added to the candidate list (called **candidate split**)
4. sort the candidate list along two dimensions: first, ascending along the candidates' number of words; second, alphabetically ascending (e.g., *material* is sequenced before *batch size* in the candidate list).

Finally, a comprehensive list of unique terms is provided for the term validation. The terms can be validated in two different ways, full and minimal (see Section 4.3.4.2).

#### 4.3.4.2 Term Validation (semi-automated)

In a next step, a domain expert validates the candidate list in order to explicitly determine true domain terms. The domain terms are particularly used by SOFEX to identify potentially software feature-relevant sentences. Evaluations show that the identification does not necessarily require a complete domain terminology (see Sections 5.3.2). Thus, in context of SOFEX, two different domain term validation strategies are provided, namely a minimal and a full candidate validation (see Figure 4.6). Following the **full candidate validation**, a domain expert requires to validate each single candidate of the candidate list. In order to “ease” the validation, the candidates are provided to the domain expert in a context-sensitive order: after validating a specific term<sub>i</sub> (e.g., “material”), the domain expert next needs to validate all terms which contain term<sub>i</sub> (e.g., “batch material”, “material list”). As an example, in Figure 4.6(a), the domain expert defines “batch” to be a domain term (see ①). The subsequent candidate in the candidate list is “box”. But, in order to remain in the context of the validated term “batch”, the domain expert is now provided all candidates which contain “batch” (“batch size” and “material batch”, see ② and ③). After the domain expert validates each of the context-sensitive candidates, the domain expert proceeds in validating the initially subsequent candidates (e.g., “box”, see ③). The output of a full candidate validation is called a full domain terminology (**FT**) and represents a complete domain terminology.

On the other hand, a **minimal candidate validation** automatically “skips” context-

sensitive candidates and thus decreases the number of domain terms to be validated by the domain expert. Thus, if, e.g., *material* is already considered to be a domain term, all other terms to be validated which contain the word *material* (e.g., *material list*) are considered to be irrelevant and thus non-domain terms in the course of a minimal candidate validation. The idea is that if, e.g., a sentence which contains the term *material list* will anyway be identified to potentially contain software feature-relevant information because it contains the word *material*. The advantage of a minimal candidate validation in contrast to a full candidate validation is that the validation effort decreases by about 20% (see Section 5.3.4). As an example, in Figure 4.6(b), the domain expert defines “batch” to be a domain term (see ①). In contrast to a full candidate validation, the context-sensitive candidates (“batch size” and “material batch”) are automatically filtered (considered to be not domain specific) from the candidate list (see ②) and the domain expert is provided the subsequent candidates in the list (“box”, see ③). In general, it is important to consider that a candidate which contains a domain term is not necessarily considered to be a domain term too (e.g., *material* is a domain term, *material view* is not a domain term, according to the domain terminology gold standard). The output of a minimal candidate validation is called a minimal domain terminology (MT) and does typically not represent a complete domain terminology.

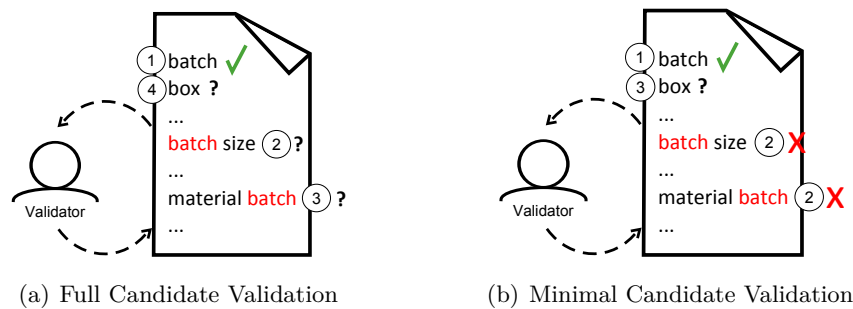


Figure 4.6: Different Types of Terminology Candidate Validation

On the one hand, creating a full domain terminology is more time consuming compared to creating a minimal domain terminology, but a full domain terminology provides detailed domain insights in a specific domain as it usually captures more domain terms than a minimal domain terminology. On the other hand, creating a minimal domain terminology requires less manual effort. For a detailed evaluation see Section 5.3.2.


#### 4.3.5 Software Feature-relevant Sentence Identification (automated)

Software Feature-relevant Sentence Identification aims to identify sentences which potentially contain software feature-relevant information by means of a domain terminology.

Similar to P7, SOFEX determines all sentences which contain at least one domain term. These sentences are then considered to potentially contain software feature-relevant information. Vice versa, all the other sentences which do not contain a domain term are considered software feature-irrelevant and are not used for further analysis and processing (see (A) in Figure 4.7)

[Section] ~~1 Quantification of Materials~~ (5)

[Section] ~~1.1 Run the GDC Quantificator~~ (5)

 GDC performs 3 main steps in the quantification process. The quantification process is based on the workload entered and on the settings made on customer environment level.

The Quantificator calculates

[Bullet] 1. the materials which are required to perform the entered number of reportables or tests


[Bullet] 2. the optimal pack sizes of each material



[Bullet] 3. the boxes for pricing parameters (~~← configuration frames~~) (1)

[Text] The result of the quantification process is a material list. The material list contains materials required to run the tests.


The quantificator runs automatically before the material list or the box configuration window is displayed or the price finding process runs.

Click the "Quantificate All" menu item from the "tools" menu to start the quantificator manually.

A red bubble  ahead of a system indicates that the quantificator needs to run (~~see the picture below~~) (1) (3)

  (1) (3)

[Step] ~~Click the "Quantificate All" menu item from the "Tools" menu.~~ (5)

 **RESULT** (5)

[Result] ~~Each row in the materials grid represents an object assigned to the level determined by the selected list view.~~

[Text] Following values are available as calculation base:

[Bullet]  Once

[Bullet Text] ~~non annual quantity = quantity \* PIE common product record~~ (4)

[Text] ~~On a lower level, GDC rounds quantities to integers only. E.g., if in the example above, the calibrator was replaced by an autopull, GDC would assign one pack to test 1 and two packs to test 2 (or vice versa), but not 1.5 packs to each test.~~ (2)

[Text] ~~Errors and warnings are displayed in a popup window.~~ (A)

Figure 4.7: Information Filtering Example

Figur 4.7 shows an excerpt of the GDC user manual and filtered sentences and phrases. The grey shaded rectangles indicate the sentence type of a corresponding sentence. In that context, the following five **exclusion patterns** which allow to automatically determine software feature-irrelevant sentences or phrases in sentences were uncovered in the course of GDC user manual investigations:

1. phrases in brackets (see (1) in Figure 4.7)
2. phrases or sentences starting with "e.g." (see (2) in Figure 4.7)
3. phrases or sentences including at least one of the domain specific stop words



“section”, “figure”, “example”, “table”, “chapter”, or “picture” (see ③ in Figure 4.7)

4. phrases or sentences which represent formulas (see ④ in Figure 4.7)
5. sentences of the types section, result, and step (see ⑤ in Figure 4.7).

The output of the Software Feature-relevant Sentence Identification is a set of potentially software feature-relevant sentences which serves as the basis to extract atomic software feature-relevant information in the next step.

## 4.4 Information Extraction (automated)

SoFEX uses information extraction in two different contexts, namely (a) the extraction of domain terms from the user manual (see Section 4.3.4) in order to identify potentially software feature-relevant sentences and (b) the extraction of atomic software feature-relevant information from the software feature-relevant sentences. According to R.3, atomic software feature-relevant information has to be as simple as possible. “Simple” in context of sentences refers to the complexity of transmitted thoughts with a single sentence: an atomic (simple) sentence conveys a single thought or information whereas a complex sentence conveys more than one thought (e.g., combined by a conjunction “and”). Similar to P4, SoFEX applies a rule-based information simplification (see Section 4.4) which is used to reduce the complexity of sentences and furthermore separate the information included in sentences.

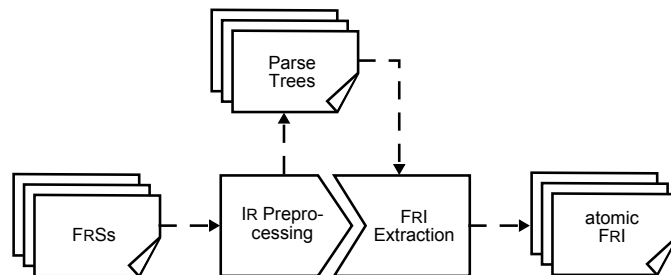


Figure 4.8: Information Extraction

Figure 4.8 shows an overview of the information extraction of SoFEX. It consists of the two steps information-related preprocessing and software feature-relevant information extraction which are explained in detail in the following two sections. Information-related preprocessing aims to provide parse trees of the potentially software feature-relevant sentences with a high accuracy to extract atomic software feature-relevant information in a next step by means of syntactic patterns.

#### 4.4.1 Information-related (IR) Preprocessing (automated)

Similar to P4, SOFEX extracts atomic software feature-relevant information based on syntactic patterns in parse trees. Thus, SOFEX highly depends on correct parse trees. Similar to TR preprocessing (see Section 4.3.3), the Stanford POS parser generates a parse tree for each sentence. SOFEX does not reuse the parse trees generated during TR preprocessing (see Section 4.3.3) because their accuracy is not sufficient enough. Thus, SOFEX applies textual modifications (see Section 4.4.1.1) on the potentially software feature-relevant sentences first. These sentences are then used by the Stanford POS parser to create parse trees. Finally, parse tree transformations (see Section 4.4.1.2) are applied on the parse trees. These measurements allow to significantly increase the parse tree accuracy (see Section 5.2.2).

##### 4.4.1.1 Terminology-based textual modifications (automated)

From a syntactical point of view, each domain term represents a (complex) noun. Thus, the domain terms in the potentially software feature-relevant sentences are indicated (e.g., “material list” → “DTmaterialDTlist”), similar to the quoted phrases in course of the lexical-based textual modifications during TR preprocessing (see Section 4.3.4). The consequence is that the POS parser treats the indicated terms as single nouns instead of trying to parse the terms separately (see, e.g., Ghosh et al., 2014). As an example, Figure 4.9 shows the difference of the parse trees: Figure 4.9(a) treats the domain term “gdc” as verb (blue rim) which is actually incorrect and furthermore distorts the entire parse tree structure. Figure 4.9(b) shows the same sentence with the lexical modifications applied. Here, “DTgdc” is correctly treated as a noun (blue rim) and thus the entire parse tree becomes correct.

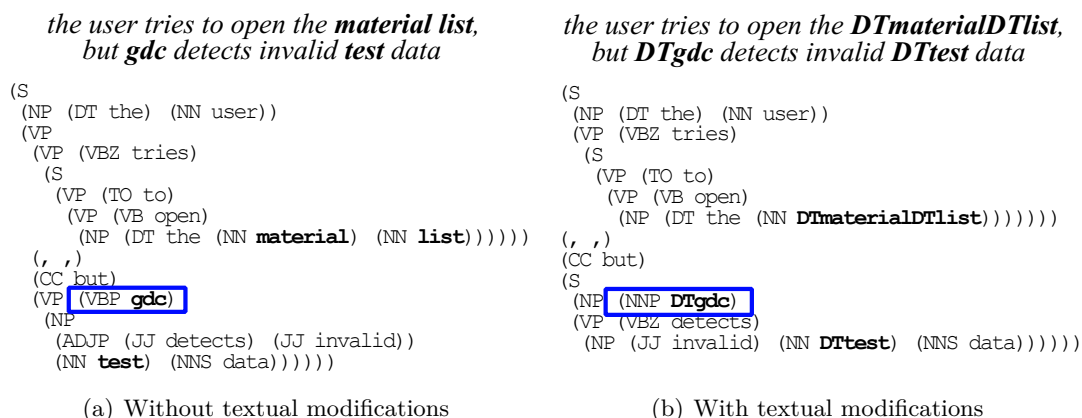


Figure 4.9: Parse tree accuracy increases with domain term (**bold**) bundling

#### 4.4.1.2 Pattern-based parse tree transformations (automated)

Parse tree transformations comprise corrections as well as adaptations to ease the extraction of atomic information afterwards. Both tasks are applied fully automated. A detailed overview of the patterns as well as corresponding transformations which are described in the following is provided in the Annex.

1. **Pattern-based parse tree corrections (automated).** In total, 17 recurring patterns (see Section B.1 in the Appendix) in parse trees are identified which indicate incorrect parts of a parse tree in context of SOFEX. By means of Tregex (Levy and Andrew 2006), which is a utility for matching patterns in parse trees, the incorrect parts of a parse tree can be identified. Tsurgeon (Levy and Andrew 2006), which is a tree-transformation utility built on top of Tregex, allows to manipulate the identified parse trees as desired. Figure 4.10 shows an example of an incorrect parse tree (left-hand side), which is corrected (right-hand side) by means of a Tregex pattern (`tregexPattern` compiled with the `patternString`) and a Tsurgeon operation (`surgery` compiled with the `surgeryString`, see upper part). The statement `Tsurgeon.processPattern` transforms each part of the parse tree `tree` which equals the pattern `tregexPattern` according to the defined operation `surgery`.

```
String patternString = "NP=par $+ (NP=del <+(NP) (NP < __=mov $+ __=mov2))";
TregexPattern tregexPattern = TregexPattern.compile(patternString);
String surgeryString = "[move mov >-1 par][move mov2 $- par][delete del]";
TsurgeonPattern surgery = Tsurgeon.parseOperation(surgeryString);
Tree tree = Tsurgeon.processPattern(tregexPattern, surgery, tree);
```

```
(ROOT
(NP
(NP (DT the) (JJ optimal) (NN pack))
(NP
(NP (NNS sizes))
(PP (IN of)
(NP (DT each) (NNP material))))))
(NP (DT the) (JJ optimal) (NN pack) (NNS sizes))
(PP (IN of)
(NP (DT each) (NNP material))))))
```

Figure 4.10: Parse Tree Transformation Example in Java

2. **Pattern-based parse tree adaptations (automated)** Furthermore, 36 patterns (see Section B.2 in the Annex) and corresponding Tsurgeon operations were identified. These patterns do not indicate wrong parts of a parse tree. They rather modify the parse trees allowing to identify LIM elements at first glance and thus ease their extraction. The applied transformations can be summarized in the following three categories:

- Condense Complex Noun Phrases. Complex noun phrases are sometimes hierarchically structured (see red rim on the left-hand side of Figure 4.11). Thus, the structured noun phrases are identified by Tregex patterns and are furthermore condensed by corresponding Tsurgeon operations (see red rim on

the left-hand side of Figure 4.11).

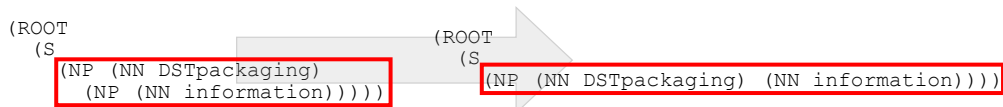


Figure 4.11: Condense Complex Noun Phrases in Parse Trees

- **Condense Complex Verb Phrases.** Similar to complex noun phrases, complex verb phrases exist. In parse trees, they are in general hierarchically structured (see red rim on the left-hand side of Figure 4.12). Therefore, the structured phrases are condensed into a single verb phrase (see red see red rim on the right-hand side of Figure 4.12).

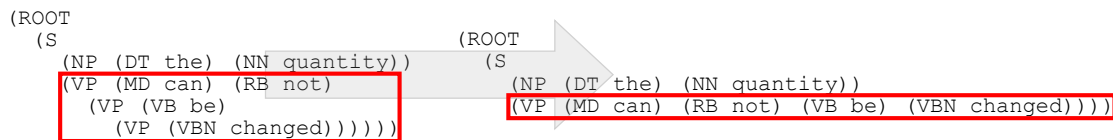


Figure 4.12: Condense Complex Verb Phrases in Parse Trees

- **Indicate LIM Entities.** Finally, parse tree transformations are applied which allow to indicate complements (e.g., *preposition complement*), modifiers, and clauses (e.g., *To Clause*) according to the LIM model (see Figure 2.2 in Section 2.3), in order to ease their extraction. Therefore, several individual POS tags are defined:
  - **NPp** preposition phrase as a noun phrase modifier (e.g., material *of*...),
  - **NPv** verb phrase as a noun phrase modifier (e.g., amount *entered*...),
  - **NPw** wh clause as a noun phrase modifier (e.g., materials *which*...),
  - **NPT** to clause as a noun phrase modifier (e.g., alternative way *to create*...),
  - **PPN** noun phrase as a preposition complement (e.g., *of the material*...),
  - **PPv** verb phrase as a preposition complement (e.g., *by using*...),
  - **PPw** wh clause as a preposition phrase complement (e.g., *for which*...),
  - **VPv** verb phrase as a verb phrase complement (e.g., objects *based on*...),
  - **VPp** preposition phrase as a verb phrase complement (e.g., add material *by*...),
  - **VPH** that clause as a verb phrase complement (e.g., configure *that*...),
  - **VPT** to clause as verb complement (e.g., start *to extract*...),
  - **VPC** condition phrase as a verb phrase complement (e.g., ends *when*...),
  - **VPw** wh clause as a verb phrase complement (e.g., changes *which* allows

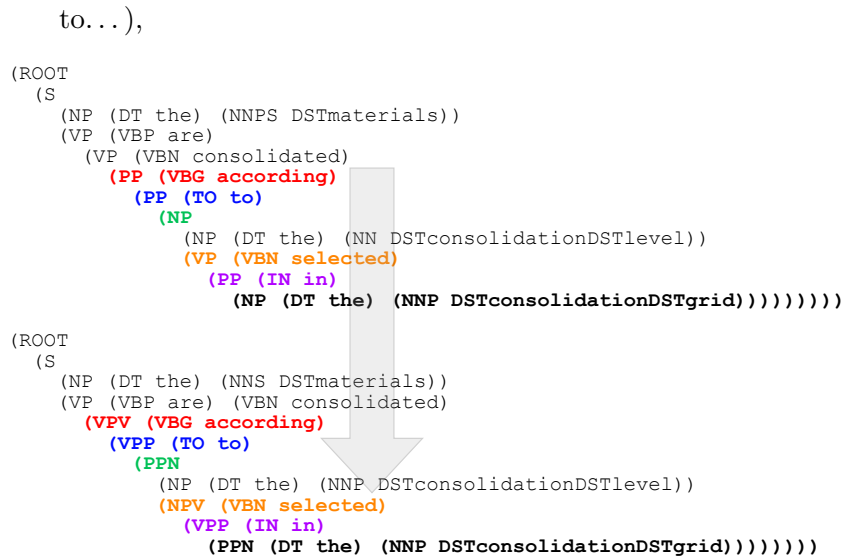


Figure 4.13: Indicate LIM Complements and Modifiers in Parse Trees

The example in Figure 4.13 depicts some of the aforementioned parse tree transformations related to the indication of LIM entities. The upper parse tree depicts the result of the Stanford PoS parser. In contrast, the lower parse tree results from the application of the parse tree transformations. The different colors indicate the separate transformations applied (e.g., green: NP is transformed to PPN).

#### 4.4.2 Software Feature-relevant Information Extraction (automated)

After successfully identifying potentially software feature-relevant sentences and adapting their parse trees, atomic software feature-relevant information is extracted in four iterative steps by means of syntactic PoS patterns.

1. **Syntactic Information Extraction (automated)**. In a first step, potentially software feature-relevant information is extracted from each sentence by traversing the parse tree and determining relevant LIM elements iteratively by means of PoS patterns (e.g., a preposition following a noun phrase like “number of reportables” indicates a modifier of the type preposition phrase). On top-level, the following LIM elements are extracted:

- a) Subject (noun phrases + modifiers): a subject is defined to be represented by a noun phrase (NP) in a sentence which is dominated by Root via an unbroken chain (>+) of simple declarative clauses (S). (NP >+(S) ROOT). In Figure 4.13, the subject is the first noun phrase (NP (DT the) (NNPS DSTmaterials)). Similarly, after the extraction of the noun phrase, possible noun phrase

modifiers are identified and extracted by means of Tregex patterns.

- b) Predicate (verb phrases + complements): a predicate is defined to be represented by a verb phrase in a sentence which is dominated by the **ROOT** via an unbroken chain of **S** nodes (**VP** >+(**S**) **ROOT**). In Figure 4.13, the predicate is the first verb phrase (**VP** (**VBP** *are*) (**VBN** *consolidated*)). Similarly, after the extraction of the verb phrase, possible verb phrase complements are identified and extracted by means of Tregex patterns.
- c) Objects of predicates (noun phrases + modifiers): an object is defined to be represented by a noun phrase in a sentence which is immediately dominated by the predicate (**NP** > /%**s**/, where %**s** represents the predicate).
- d) Conditional clauses: they may occur on top-level of a sentence (e.g., “If the quantificator runs,...”) and are identified with the pattern **SBAR|VPC** [< **IN** | < **WHADVP**] >+(**S|VPC|SBAR**) **ROOT**. It defines a conditional clause (**SBAR|VPC**) to be immediately followed by a preposition (**IN**) or a Wh-adverb phrase (**WHADVP**) and is dominated via an unbroken chain (>+) of simple declarative clauses (**S**), conditional phrases as verb phrase complements (**VPC**), or clauses introduced by a subordinating conjunction (**SBAR**).
- e) Preposition phrase: similar to conditional clauses, preposition phrases may occur on top-level of a sentence (e.g., “In the ‘dates’ window,...”) and can be identified with the pattern **PP** >+(**S**) **ROOT**. It defines a preposition phrase (**PP**) on top-level to be dominated by **ROOT** via an unbroken chain (>+) of simple declarative clauses (**S**).

Each potential software feature-relevant sentence results in exactly one unit of **potential** software feature-relevant information which might not be atomic yet.

2. **Information simplification (automated)**. In order to retrieve atomic units of information, all conjunctions in phrases, clauses (except conditional clauses), complements, and modifiers are resolved (see P4); each conjunction element becomes part of a new atomic unit of information (see Figure 4.14). Conjunctions are determined by means of the PoS tag **CC**.
3. **Enumeration Resolution (automated)**. The user manual excerpt on the left-hand side of Figure 4.15 shows a paragraph with a bullet list. Paragraphs which contain bullet lists need to be resolved in order to retrieve atomic information (see P4). Resolving a bullet list means to combine the potential software feature-relevant information from the introducing sentence (e.g., *The quantificator calculates*) with each potential software feature-relevant information of the related bulleted sentence (e.g., *the optimal pack size of each material*), depending on the corresponding syntax of the information (see Figure 4.15).

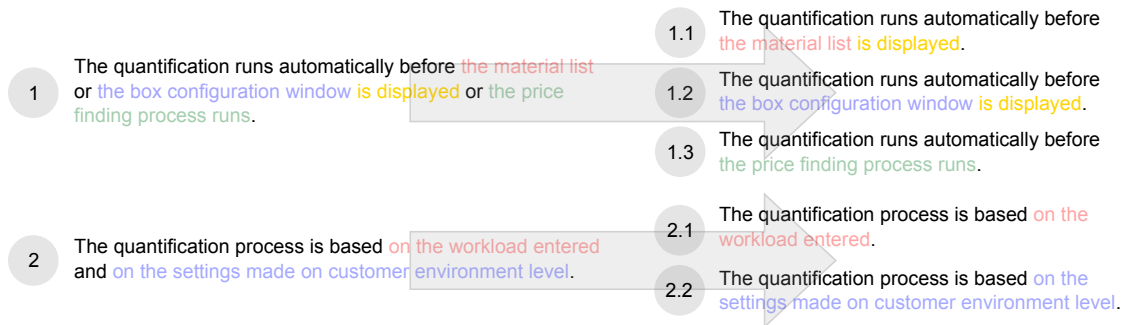


Figure 4.14: Information Simplification Example

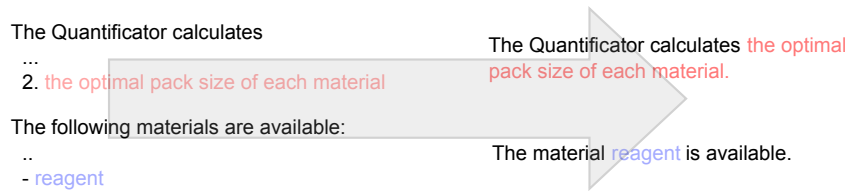


Figure 4.15: Enumeration Resolution Example

In total, 8 different syntactical patterns based on LIM elements are identified which entail the combination of an initiating sentence (black sentences in Figure 4.15) and a contextually related bulleted sentences (red and blue sentences in Figure 4.15). If one of the following patterns is found in an initial sentence (Is), the bulleted sentences (BS) are appended to the initial sentences' subject or predicate:

- a) append BS as object of Is.**predicate** (e.g., Is: “the quantificator calculates”, BS: “boxes for pricing” → “the quantificator calculates boxes for pricing”)
- b) append BS as (additional) complement of Is.**predicate** where no object of Is.**predicate** exists (e.g., textscIs: “the quantity cannot be set to 0”, BS: “for mandatory associated items” → “the quantity cannot be set to 0 for mandatory associated items”)
- c) append BS as (additional) complement of Is.**predicate** where an object of Is.**predicate** exists (e.g., Is: “exclude optional ccc materials”, BS: “on deal level” → “exclude optional ccc materials on deal level”)
- d) integrate BS into Is.**subject** (e.g., Is: “the following values are available as calculation base”, BS: “batch size” → “the value batch size is available as calculation base”)
- e) integrate BS in Is.**predicate.object** (e.g., Is: “the column XY accepts the following values”, BS: “partly edited” → “the column XY accepts the value partly edited”)
- f) append BS as (additional) modifier of Is.**predicate.object** (e.g., Is: “the

customer role defines the credentials ”, Bs: “of a primary pricing customer”  
 → “the customer role defines the credentials of a primary pricing customer”)

g) integrate Bs in `Is.predicate.object.complement` (e.g., Is: “the user can collapse the environment list along the following levels”, Bs: “test ” → “the user can collapse the environment list along the level test”)

h) integrate Bs in `Is.predicate.object` (e.g., Is: “the context menu offers additional functionality as follows”, Bs: “’open log file’ ” → “the context menu offers additional functionality ’open log file’ ”)

4. **Syntactic Relevance Determination (automated)**. Finally, software feature-relevant and software feature-irrelevant information is distinguished based on the LIM syntax. A unit of information is syntactically relevant (and thus software feature-relevant) if its syntax (based on the LIM elements) equals one out of 11 predefined patterns; else it is considered syntactically irrelevant and thus software feature-irrelevant. A pattern defines the LIM-elements (e.g., subject + predicate + object) which need to be present in an atomic software feature-relevant unit of information to be considered truly software feature-relevant (e.g., *the quantifier calculates materials.*) On the other hand, each potentially software feature-relevant information which does not match a predefined pattern (e.g., “*The quantifier runs.*” contains only subject + predicate) is considered software feature-irrelevant. Table 4.1 provides an overview of the different patterns which are considered syntactical relevant.

Table 4.1: Syntactical Relevancy Patterns

Pattern	Example
S - [(P - O)   P - CC]	“quantifier starts quantification”, “quantifier starts if..”
S - (P - TH)	“quantification indicates that..”
S - (P - TO - O)	“quantifier starts to run quantification”
S - (P - TO - CC)	“quantifier starts to run if..”
S - (P - PP - NP)	“quantification can be started by user”
S - (P - (PP - VP - O))	“level is selectable by using views”
S - (P - (PP - NP))	“materials are used according to quantification”
S - (P - ADJ)	“levels are available”
S - (P - CC)	“levels indicate whether..”
S - P - PP	“changes are reset by ...”
P - O - TO	“start quantifier to ...” (any predicate complement)

S...subject, P...predicate, O...object, CC...condition clause, TH...that clause,  
 TO...to clause, PP...preposition phrase, VP...verb phrase, NP...noun phrase,  
 ADJ...adjective



## 4.5 Information Assignment (semi-automated)

In a last step, SOFEX aims to use the extracted atomic software feature-relevant information in order to describe software features in detail. The initial idea was to cluster the extracted atomic software feature-relevant information to software features in a fully automated and unsupervised manner. Evaluations showed that it is not possible to automatically uncover software features from scratch by means of clustering or topic modeling algorithms with high precision and recall (see Section 5.2.4). Thus, SOFEX uses a classification algorithm to recommend those software features to domain experts which most likely appear to serve as the appropriate and therefore logically related software feature to assign the extracted atomic FRI (see Figure 4.16). Information assignment in SOFEX consists of the three steps assignment-related (AR) preprocessing, learning, and software feature knowledge enhancement which are described in the following sections.

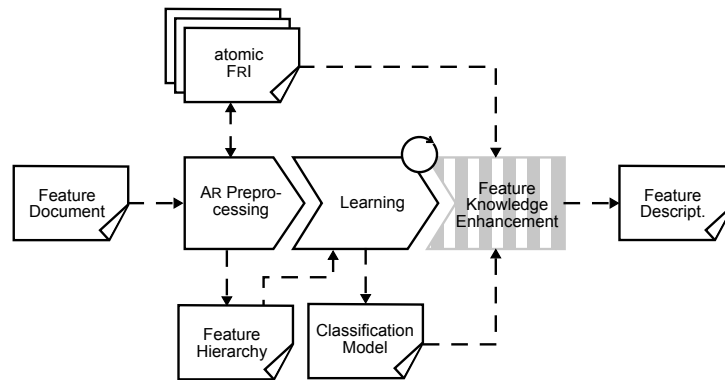


Figure 4.16: Information Assignment Process

### 4.5.1 Assignment-related (AR) Preprocessing (automated)

Text classification applications typically use preprocessing in order to normalize text data. In that context, normalization aims to remove noise and standardize words to its base forms. Thus, SOFEX applies several subsequential steps to preprocess the textual entities which are used for classification. The textual entities used in Information Assignment is a set of atomic software feature-relevant information which needs to be assigned to the corresponding software features derived from the Feature Document (provided by Roche). The Feature Document contains a structured overview of all the software features from GDC with a short descriptions (see Figure 4.18).

The feature hierarchy represents the skeleton of the feature description. Figure 4.17 depicts the meta model of the Feature Hierarchy and Feature Description, respectively. The meta model is based on the composite design pattern (see, e.g., Gamma, 1995)

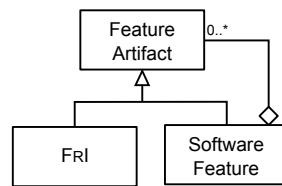


Figure 4.17: Feature Hierarchy and Feature Description Meta Model

and defines a software feature to consist of another (sub-)software feature or an atomic software feature-relevant information (FRI). Both, software feature and atomic software feature-relevant information are subtypes of the abstract type *feature artifact*.

Similar to document preparation (see Section 4.3.2), the Apache POI API is used to extract the NL text from the Feature Document in the form of a \*.doc and transform it into an internal Feature Hierarchy which is similar to the Dataset (see Section 4.3.2). Each heading in the Feature Document is transformed to a software feature and each sentence is transformed to an atomic software feature-relevant information in the Feature Hierarchy and assigned to its corresponding software feature (see Figure 4.18).

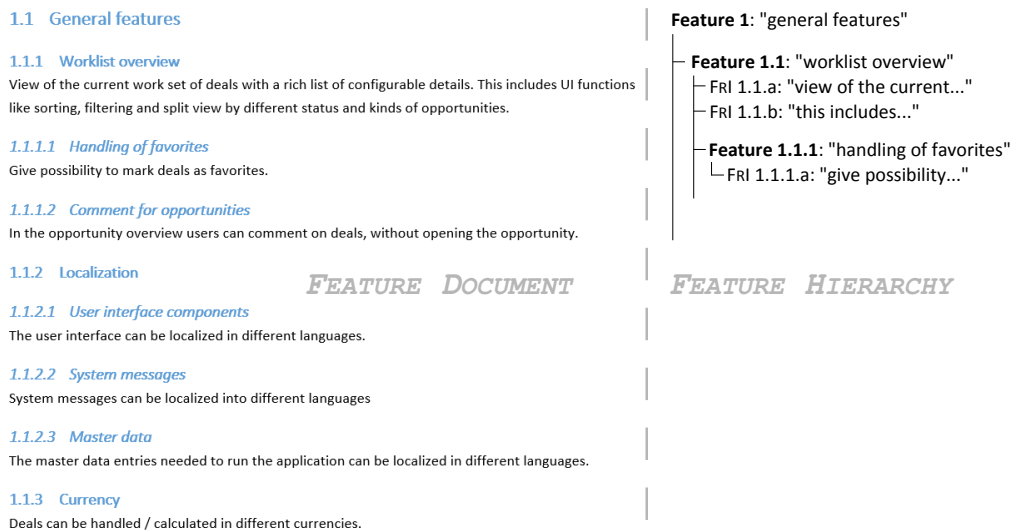


Figure 4.18: Roche’s GDC Feature Document Transformation

While transforming the NL text of the Feature Document into the Feature Hierarchy, several preprocessing tasks are applied. First, lowercasing transforms all text data to lowercase characters. Second, regular expressions are used to remove non-letters (e.g., punctuation, numbers, arithmetic signs). Furthermore, by means of a generic stoplist (see Section A.2 in the Annex), stop words are removed. Finally, the remaining words are stemmed and lemmatized. As an example, the original text in the Feature Document from Figure 4.18 *View of the current work set of deals with a rich list of configurable*

*details*“ is transformed to *”view cur work se deal rich list configur detail“* in the Feature Hierarchy.

#### 4.5.2 Learning (automated)

Basically, most classification algorithms cannot handle text data. If they can do it, their running time is usually much longer compared to classification algorithms which use numeric data (see, e.g., Hassan et al., 2011). Thus, SOFEX transforms the textual representation of the text data into a VSM representation (see Section 2.6.2) before training the classifier. The classifier initially uses the atomic software feature-relevant information from the feature hierarchy for learning purpose: each feature in the feature hierarchy represents a *”class“* in the classifier and the related atomic software feature-relevant information is used as training data for that specific class. The output of the learning step is a trained classification model which serves as the basis for classification. SOFEX uses Weka for classification tasks.

```
@attribute class {1,1.1,1.1.1,1.1.2,...}

@attribute view numeric
@attribute cur numeric
@attribute work numeric
@attribute se numeric
@attribute deal numeric
@attribute rich numeric
@attribute list numeric
@attribute configur numeric
@attribute detail numeric

@data
{0 1.1,1 1,2 1,3 1,4 1,5 1,6 1,7 1}
```

Figure 4.19: Exemplary Classification Model (Weka ARFF File)

Weka uses so called ARFF files to persist a classification model. An example of an ARFF file is given in Figure 4.19. attribute declarations are represented as an ordered sequence of **@attribute** statements. Each attribute statement uniquely defines the name of that attribute (e.g., *”configur“*) and its data type, in that case *”numeric“*. Each distinct and lemmatized word from each atomic software feature-relevant information which was used for training purpose is captured in an attribute statement. The order of the attributes defines the attribute index which is used in the data section (**@data**). **Class** is a predefined attribute and represents a set of all possible classes (in case of SOFEX the index of the features, see right hand side of Figure 4.18) a record can be classified as. Each record in the data section corresponds to a single atomic software feature-relevant information which was used for training purpose. The first data pair in the record’s set (*”0 1.1“*) specifies the value of the attribute with the index 0 (class). Thus, the record defines a training record for the feature *”worklist overview“* (see right hand

side of Figure 4.18). The subsequent data pairs (e.g., "1 1") indicate which attributes are contained in the corresponding record. Related to the exemplary atomic software feature-relevant information from Section 4.5.1 ("view cur work se deal rich list configur detail"), the record "1 1" defines that attribute 1 ("view") occurs once in the record; the record "2 1" defines that attribute 2 ("view") occurs once in the record, and so on.

### 4.5.3 Software Feature Knowledge Enhancement (manual)

In the last phase, the set of atomic software feature-relevant information is assigned to the corresponding software feature in a semi-automated way. Basically, a domain expert has to assign each atomic atomic software feature-relevant information to the desired software feature in a manual way. Figure 4.20 shows the process of software feature knowledge enhancement.

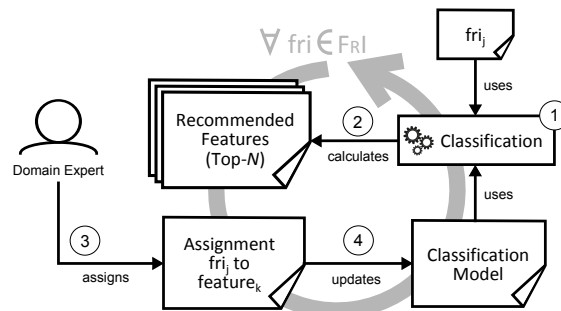


Figure 4.20: Feature Knowledge Enhancement Process

In context of GDC, the feature hierarchy contains 373 features. Thus, a manual assignment of a specific atomic software feature-relevant information without automated support is very cumbersome and time consuming (which was observed by domain experts during the creation of the feature-information assignment gold standard, see Section 5.2). Therefore, SOFEX uses a classification algorithm which determines the likelihoods of a logical relationship of an atomic software feature-relevant information  $fri_j \in FRI$  to each software feature (see ① in Figure 4.20). The likelihoods are then sorted in descending order, and the top- $N$  features are provided to the domain expert (see ② in Figure 4.20). Nevertheless, the domain expert needs to manually choose a feature $_k$  to assign  $fri_j$  (see ③ in Figure 4.20). After assigning  $fri_j$  to feature $_k$ , that assignment is used to update the classification model (see ④ in Figure 4.20). The evaluation of Information Assignment (see Section 5.2.4) compares an updateable version of the Naive Bayes classifier and the SVM classifier which requires the entire classification model to be trained from scratch every time new training data needs to update the classifier.

## Part IV Treatment Validation

# Evaluation

This chapter contains the evaluations of SOFEX. Section 5.1 introduces the evaluation environment as well as the research questions to be answered in context of the evaluations. Basically, there are two different evaluations: the first (Section 5.2) uses the domain terminology gold standards for the application of SOFEX, the second (Section 5.3) uses the semi-automatically extracted domain terminologies from domain experts for the application of SOFEX.

## 5.1 Introduction

The GDC user manual contains more than 700 pages in its current version. The two example sections chosen for evaluation comprise in total 50 pages. The sections contain 1161 complete as well as incomplete sentences. The sentences are distributed over the different sentence types (see Section 4.3.2) as follows: 46.7% text (543 sentences), 29.6% bullet text (343 sentences), 7.9% bullet (92 sentences), 5.2% step (60 sentences), 3.9% section (45 sentences), 3.5% result (41 sentences), 2.3% sub-bullet text (27 sentences), and 0.9% sub-bullet (10 sentences).

Generally, in order to evaluate the process steps of SOFEX, the following five gold standards are used:

- minimal domain terminology ( $\mathbf{MT}_G$ ) and full domain terminology ( $\mathbf{FT}_G$ )
- software feature-relevant sentences ( $\mathbf{FS}_G$ )
- parse trees ( $\mathbf{PT}_G$ )
- software feature-relevant information ( $\mathbf{FI}_G$ )
- software feature-information assignment ( $\mathbf{AS}_G$ ).

The gold standards  $\mathbf{FT}_G$ ,  $\mathbf{FS}_G$ , and  $\mathbf{FI}_G$  were created by collaborating GDC experts with

long-standing expertise in the field of GDC. Each sentence of the user manual excerpt was investigated and determined whether or not it contains software feature-relevant information and domain terms:  $\mathbf{FS}_G$  contains 639 software feature-relevant and 521 software feature-irrelevant sentences,  $\mathbf{FT}_G$  contains 119 domain terms, and  $\mathbf{MT}_G$  contains 80 domain terms.  $\mathbf{MT}_G$  was automatically created as a subset of  $\mathbf{FT}_G$  ( $\mathbf{MT}_G \subset \mathbf{FT}_G$ ): each domain term (n-gram:  $n \geq 2$ ) in  $\mathbf{FT}_G$  which contains another domain term (m-gram:  $m < n$ ) is not considered in  $\mathbf{MT}_G$ . As an example, if *material* and *material list* are domain terms in  $\mathbf{FT}_G$ , *material* is considered and *material list* is not considered as a domain term in  $\mathbf{MT}_G$ . Nevertheless, a domain term in  $\mathbf{FT}_G$  which is not included in  $\mathbf{MT}_G$  contains at least one word which is already a domain term in  $\mathbf{MT}_G$ . Furthermore, the GDC experts extracted the atomic software feature-relevant information from the 631 software feature-relevant sentences. In total, they determined 849 atomic software feature-relevant units of information which are contained in  $\mathbf{FI}_G$  which resembles 1.36 software feature-relevant units of information per software feature-relevant sentence. Each atomic software feature-relevant unit of information was then assigned by the GDC experts to its corresponding software feature which resulted in the  $\mathbf{AS}_G$ : the 849 atomic software feature-relevant units of information were assigned to 73 out of 373 software features, resembling 11.63 software feature-relevant units of information per software feature.  $\mathbf{PT}_G$  was provided by a non-domain expert as there is no GDC-specific knowledge required.

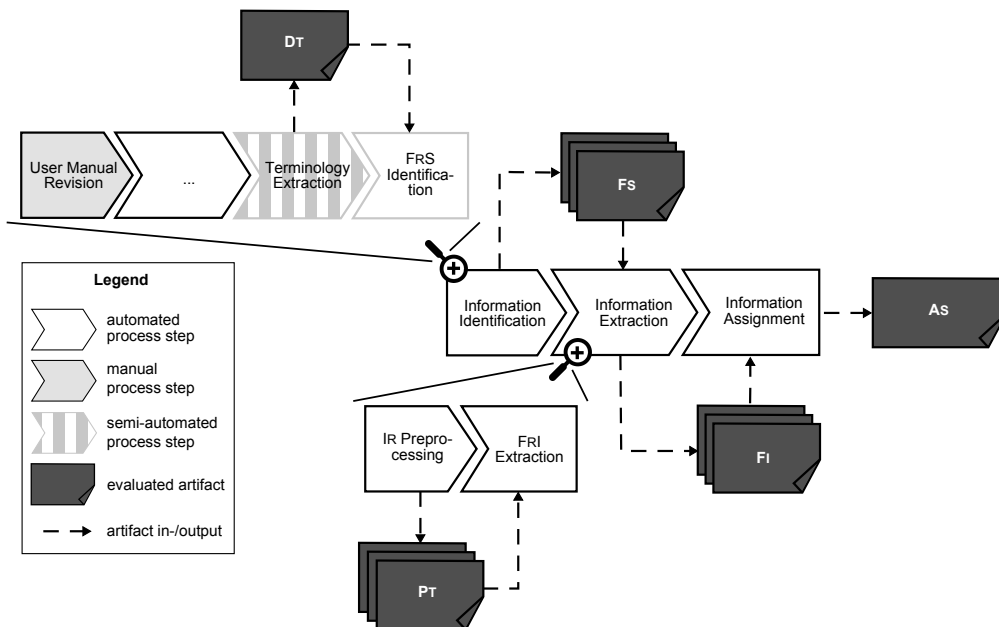


Figure 5.1: SoFeX Evaluation Overview

Figure 5.1 shows an excerpt of the process steps and corresponding created artifacts of

SoFEX which are evaluated:

- in context of Terminology Extraction, the extracted domain terminologies (DT) are evaluated in relation to  $\mathbf{FT}_G$  and  $\mathbf{MT}_G$  (see Section 5.3.1).
- in context of Information Identification the identified software feature-relevant sentences (Fs) are evaluated in relation to  $\mathbf{FS}_G$  (see Sections 5.2.1 and 5.3.2).
- in context of IR Preprocessing, the created parse trees (PT) are evaluated in relation to  $\mathbf{PT}_G$  (see Sections 5.2.2 and 5.3.3).
- in context of Information Extraction, the extracted set of software feature-relevant information (FI) is evaluated in relation to  $\mathbf{FI}_G$  (see Sections 5.2.3 and 5.3.4).
- in context of Information Assignment, the recommended assignment of software feature-relevant information to corresponding software features (AS) is evaluated in relation to  $\mathbf{AS}_G$  (see Sections 5.2.4 and 5.3.5).

Table 5.1: Different Evaluations of SoFEX

(a) Evaluation 1 (Section 5.2)

Process Step	Output	Gold Standard	Section
Information Identification	software feature-relevant sentences (Fs)	$\mathbf{FS}_G$	5.2.1
IR Preprocessing	parse trees of Fs (PT)	$\mathbf{PT}_G$	5.2.2
Information Extraction	atomic software feature-relevant information (FI)	$\mathbf{FI}_G$	5.2.3
Information Assignment	Feature Description (AS)	$\mathbf{AS}_G$	5.2.4

(b) Evaluation 2 (Section 5.3)

Process Step	Output	Gold Standard	Section
Terminology Extraction	domain terminologies ( $\mathbf{FT}_n$ , $\mathbf{MT}_n$ )	$\mathbf{MT}_G$ , $\mathbf{FT}_G$	5.3.1
Information Identification	software feature-relevant sentences (Fs)	$\mathbf{FS}_G$	5.3.2
IR Preprocessing	parse trees of Fs (PT)	$\mathbf{PT}_G$	5.3.3
Information Extraction	atomic software feature-relevant information (FI)	$\mathbf{FI}_G$	5.3.4
Information Assignment	Feature Description	$\mathbf{AS}_G$	5.3.5

Overall, the evaluations investigate two main aspects. The first evaluation (see Section 5.2) aims to determine to which degree SoFEX is able to extract software feature-relevant information under ideal conditions (by using  $\mathbf{FT}_G$  and  $\mathbf{MT}_G$ ). Table 5.1(a) provides an overview of the different process steps of SoFEX which are evaluated in context of the first aspect, their produced output, the gold standard the output is compared to, and the corresponding section of the evaluation. Besides the three main building blocks of SoFEX (Information Identification, Information Extraction, and Information Assignment; see Figure 4.1 in Section 4.2), IR Preprocessing is evaluated. Information Identification and



IR Preprocessing use the domain terminology gold standards as input ( $\mathbf{FT}_G$  and  $\mathbf{FT}_G$ ).

Based on the first evaluation, the second evaluation (see Section 5.3) aims to determine if the usage of domain terminologies which are semi-automatically extracted by the domain experts ( $\mathbf{FT}_N$  and  $\mathbf{MT}_N$ , where  $n$  equals to the corresponding domain expert) impact the results of the software feature-relevant information extraction of SOFEX. Similar to Table 5.1(a), Table 5.1(b) provides an overview of the second evaluation. In contrast to the first evaluation, the second evaluation is based on  $\mathbf{FT}_N$  and  $\mathbf{MT}_N$ : Information Identification and IR Preprocessing use  $\mathbf{FT}_N$  and  $\mathbf{MT}_N$ . Additionally to the first evaluation, the second evaluation aims to evaluate Terminology Extraction and thus  $\mathbf{FT}_N$  and  $\mathbf{MT}_N$  in relation to  $\mathbf{FT}_G$  and  $\mathbf{MT}_G$ . Both the first and the second evaluation provide results for the application of SOFEX with minimal as well as full terminologies in order to investigate the impact of different domain terminologies on the extraction results. Thus, the evaluations aim to answer the following two primary research questions in the subsequent sections:

- Q.1 To which degree, namely precision, recall,  $F_1$ -score does SOFEX allow to semi-automatically extract atomic software feature-relevant information from natural language user manuals?
- Q.2 To which degree do semi-automatically extracted domain terminologies impact the quality of atomic software feature-relevant information extraction?

All the evaluations were performed on a computer with an Intel Core i5-4300U CPU, 12 GB of RAM and a x64 Windows 8.1.

## 5.2 Evaluation I

The first evaluation aims to answer Q.1 and therefore uses the domain terminology gold standards. Figure 5.2 shows a detailed overview of the different evaluations in context of SOFEX based on the full and a minimal domain terminology gold standards  $\mathbf{MT}_G$  and  $\mathbf{FT}_G$ . The column *Process Step* depicts the selected key steps of SOFEX which are evaluated: Information Identification (see Section 4.3), IR Preprocessing (see Section 4.4.1), Information Extraction (see Section 4.4.2), and Information Assignment (see Section 4.5). The column *Evaluation Setup* shows the setups (input/output) for the different process steps. With the given input (e.g.,  $\mathbf{PT}_{TP+MT}$ ), the process steps (e.g.,  $\mathbf{IX}_{R+S+}$ ) generate output in the form of *result sets* (e.g.,  $\mathbf{FI}_{RE+MT}$ ) which are then used for evaluation. The column *Gold Standards* on the right-hand side of the figure lists the different gold standards used for the evaluation per process step. Parse tree accuracy is used to evaluate IR Preprocessing, the standard metrics precision (P), recall (R) and  $F_\beta$ -score ( $F_\beta$ ) are used to evaluate Information Identification and Information Extraction. In that context,  $F_1$ -score is used rather than, e.g.,  $F_{10}$ , because precision is explicitly

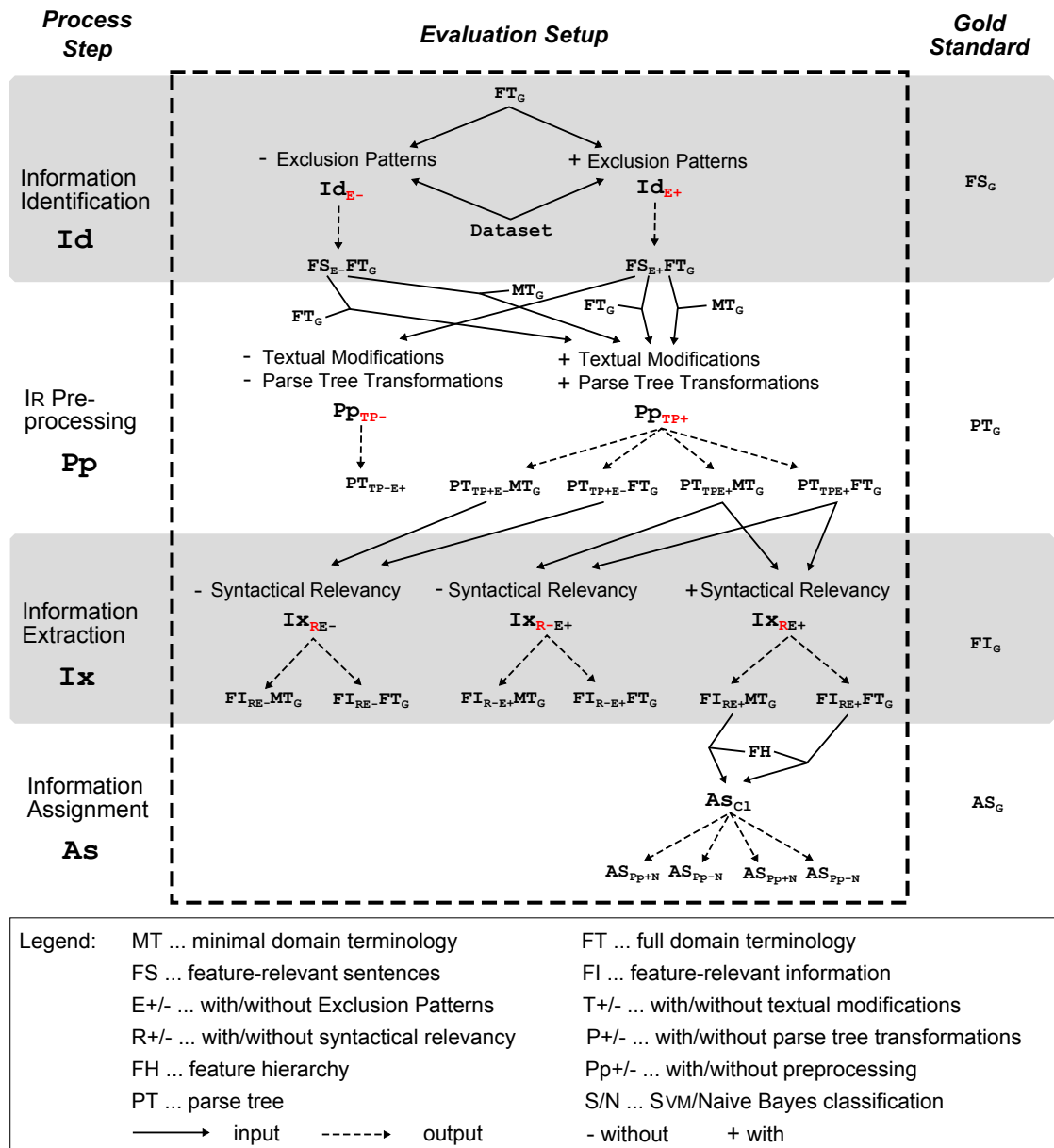


Figure 5.2: Evaluations Overview (Gold Standards)

considered as important as recall (see R.5 and R.4) in order to avoid further manual filtering. Therefore, the aim is to maximize the  $F_1$ -scores. The results of this evaluation are discussed in Section 5.2.5.

### 5.2.1 Information Identification (Id)

Information Identification uses the sentences of the *Dataset* (i.e. internal data model, see Section 4.3.2) and a domain terminology as input and determines which sentences

are potentially software feature-relevant (e.g.,  $\mathbf{FS}_{E+}\mathbf{FT}_G$ ) and which are not. As outlined in Section 4.3.4, each domain term in  $\mathbf{FT}_G$  which is not included in  $\mathbf{MT}_G$  (e.g., *material list*) contains at least one word which is already a domain term in  $\mathbf{MT}_G$  (e.g., *material*). As an example, the sentence “The *material list* shows *annual materials*” is considered potentially software feature-relevant with both  $\mathbf{FT}_G$  and  $\mathbf{MT}_G$ :  $\mathbf{FT}_G$  contains the domain terms *material list* and *annual material* which are found in the sentence,  $\mathbf{MT}_G$  contains the domain term *material* which is found twice. Thus,  $\mathbf{MT}_G$  as well as  $\mathbf{FT}_G$  identify the exact same sentences to be software feature-relevant. Therefore, only  $\mathbf{FT}_G$  is used to evaluate Information Identification. Information Identification is performed with and without considering exclusion patterns, namely  $\mathbf{ID}_{E+}$  and  $\mathbf{ID}_{E-}$ : the comparison of the results allows to determine the impact of exclusion patterns on Information Identification. As a consequence, the following questions in relation to the evaluation of Information Identification need to be answered:

Q1.1 To which degree (precision, recall,  $F_1$ -score) does SOFEX allow to semi-automatically identify software feature-relevant sentences in natural language user manuals?

Q1.2 Does the consideration of the exclusion patterns positively impact the identification of software feature-relevant sentences?

Q1.1 can be answered by interpreting the values shown in Table 5.2: it depicts that SOFEX identifies potentially software feature-relevant sentences without considering exclusion patterns ( $\mathbf{FS}_{E-}\mathbf{FT}_G$ ) with a precision and recall of 72.89% and 98.57%, respectively. This results in an  $F_1$ -score of 83.81%. Considering the exclusion patterns additionally to domain terms ( $\mathbf{FS}_{E+}\mathbf{FT}_G$ , see Section 4.3.5), precision improves to 80.69% with unchanged recall of 98.57%. Accordingly, the  $F_1$ -score increases to 88.81%. Thus, the answer to Q1.2 is that the consideration of exclusion patterns positively impacts Information Identification.

Table 5.2: Evaluation Results for Information Identification

	$\mathbf{FS}_{E-}\mathbf{FT}_G$	$\mathbf{FS}_{E+}\mathbf{FT}_G$
TP	625	625
TN	294	370
FP	227	151
FN	6	6
R	98.57%	98.57%
P	72.89%	80.69%
$F_1$	83.81%	88.81%

$\mathbf{FS}_{E[+|-]}\mathbf{FT}_G$  = set of software **F**eature-relevant **S**entences (**FS**) which was determined by using a **F**ull domain **T**erminology **G**old standard ( $\mathbf{FT}_G$ ) with (**E+**)/without (**E-**) considering **E**xclusion patterns

Additionally, clustering and classification algorithms are used to identify software feature-relevant sentences in order to compare the results with the results from SOFEX’s

Information Identification. For (unsupervised) clustering, the two popular clustering algorithms  $k$ -means (KM) and Expectation-Maximization (EM) are chosen because they allow to predetermine the number of clusters before their application (see Section 2.6.3.3): in context of Information Identification, a sentence can either be software feature-relevant or software feature-irrelevant which corresponds to exactly two clusters. Furthermore, KM and EM are widely used and well proven in the field of text clustering (see, e.g., Ghosh and Strehl, 2006). Classifying software feature-relevant sentences is performed with the two classification algorithms NB and SVM (see Section 2.6.3.2). Both NB and SVM are widely used in the field of text classification, outperform other classification algorithms (see, e.g., Colas and Brazdil, 2006), and allow an easy application. In general, classification algorithms require training data which is used to train a classification model (see Section 2.6.3). In order to evaluate the classification of software feature-relevant sentences, a standard 10-fold cross validation (see, e.g., Aggarwal and Zhai, 2012a) is performed. In a 10-fold cross validation, the text data to be classified (e.g., feature-relevant or feature-irrelevant sentences) is randomly partitioned into 10 groups of equal size. A single group is used as test data and the remaining 9 groups are used as training data which corresponds to a ratio of 9:1 for training to test data. The classification is then repeated 10 times with each of the 10 groups used exactly once as test data. Each repetition is referred to as fold. Finally, the precision, recall, and the  $F_1$ -score of the 10 folds are averaged.

	KM	EM	NB	SVM
TP	533	381	598	514
TN	59	158	313	396
FP	432	332	177	94
FN	128	281	64	148
R	55.23%	53.44%	77.16%	84.54%
P	80.64%	57.55%	90.33%	77.64%
$F_1$	65.56%	55.42%	80.94%	83.23%

Table 5.3: Evaluation of Information Identification with Clustering and Classification

Table 5.3 shows the results of clustering in grey shaded, and classifying software feature-relevant and software feature-irrelevant sentences. In general, classification outperforms clustering as shown by the higher scores for recall, precision, and  $F_1$ -score. Classification is even able to perform similar to Information Identification by using domain terms *without* considering exclusion patterns ( $\mathbf{FS}_E\text{-}\mathbf{FT}_G$  in Figure 5.2). Compared to Information Identification considering exclusion patterns (see  $\mathbf{FS}_{E+\mathbf{FT}_G}$  in Figure 5.2), however, SVM classification performs worse. Furthermore, classification is supervised and requires

training data in order to perform that well. As an example, if only the sentences from the first section of the GDC user manual (514 sentences) are used for classifier training in order to classify the sentences from the second section of the GDC user manual (638 sentences), the  $F_1$  score decreases from 83.23% to 45.03% for SVM. Here, the ratio for training and test data is about 4.5:5.5. Thus, Information Identification by means of domain terms and exclusion patterns can be considered appropriate regarding effort and performance.

### 5.2.2 IR Preprocessing (Pp)

IR Preprocessing uses domain terminologies and a set of potentially software feature-relevant sentences as input. In context of this evaluation, it is investigated to which degree the textual modifications and parse tree transformations of SoFEX impact IR Preprocessing and thus the parse tree accuracy of the generated parse trees. Therefore, IR Preprocessing was performed without textual modifications and parse tree transformations ( $PP_{TP-}$ ) and with textual modifications and parse tree transformations ( $PP_{TP+}$ ) in order to determine the impact of textual modifications and parse tree transformations on the parse tree accuracy. Without textual modifications, IR Preprocessing does not need a domain terminology as input. Additionally, this evaluation investigates to which degree the parse tree accuracy is impacted by using different domain terminologies. Thus, both  $MT_G$  and  $FT_G$  are used as input for IR Preprocessing. The evaluation is different compared to Information Identification and Information Extraction: it solely evaluates the parse tree accuracy of the generated parse trees from  $FS_G$ . In the context of IR Preprocessing, the following research question need to be answered:

- Q1.3 Is it possible to increase parse tree accuracy through automated textual modifications and parse tree transformations?
- Q1.4 Is there a difference in the parse tree accuracy if minimal or full domain terminologies are used as input for IR Preprocessing?

The results for the evaluation of IR preprocessing are depicted in Table 5.4. Without considering textual modifications and parse tree transformations, the parse tree accuracy of the generated parse trees from the 631 potentially software feature-relevant sentences ( $FS_{E+FT_G}$ ) is 82.17% using  $MT_G$  ( $PT_{TP-E+MT_G}$ ) and 83.54% using  $FT_G$  ( $PT_{TP-E+FT_G}$ ). By applying the textual modifications and parse tree transformations based on  $MT_G$ , the corresponding parse tree accuracy  $PT_{TP+MT}$  increased to 94.81%; by using  $FT_G$ , the corresponding parse tree accuracy  $PT_{TP+FT}$  increased to even 98.43%. Thus, the answer to Q1.3 is that the textual modifications and parse tree transformations allow to increase

the parse tree accuracy; in context of Q1.4, the usage of a full domain terminology for IR Preprocessing results in a higher parse tree accuracy compared to using a minimal domain terminology.

Table 5.4: Parse Tree Accuracy for IR Preprocessing

$PT_{TP-E+MT_G}$	$PT_{TP-E+FT_G}$	$PT_{TPE+MT_G}$	$PT_{TPE+FT_G}$
82.17%	83.54%	94.81%	98.43%

$PT_{TP[+|-]E+[MT|FT]_G}$  = set of **Parse Trees (PT)** generated from software feature-relevant sentences (identified by considering **Exclusion patterns - E+**) with (TP+)/without (TP-) **Textual modification** and **Parse tree transformations** by using a **Full (FT<sub>G</sub>)** or a **Minimum Terminology Gold standard (MT<sub>G</sub>)**

### 5.2.3 Information Extraction (Ix)

Information Extraction requires the identified software feature-relevant sentences' parse trees as input to determine the set of atomic software feature-relevant units of information. This evaluation aims to investigate the impact of exclusion patterns and syntactical relevancy on the recall, precision, and the  $F_1$  score of extracted atomic software feature-relevant information as output of Information Extraction. Therefore, Information Extraction is applied with and without considering exclusion patterns ( $E+|E-$ ) and syntactical relevancy ( $R+|R-$ ). Furthermore, it is evaluated if there is a difference in the recall, precision, and  $F_1$ -score of the atomic software feature-relevant information extracted from  $PT_{TPE+MT_G}$  and  $PT_{TPE+FT_G}$ . In that context, the following questions need to be answered:

- Q1.5 Does the consideration of exclusion patterns in Information Identification positively impact the recall, precision, and  $F_1$ -score of Information Extraction?
- Q1.6 Does the consideration of syntactical relevancy positively impact the recall, precision, and the  $F_1$ -score of Information Extraction?
- Q1.7 Is there a difference in the recall, precision, and  $F_1$ -score of Information Extraction by using a minimal or a full domain terminology for the identification of potentially software feature-relevant sentences?

Table 5.5 shows an overview of the different evaluations related to Information Extraction.  $F_{I_{RE-MT}}$  and  $F_{I_{RE-FT}}$  represent the sets of atomic software feature-relevant information extracted from the parse trees  $PT_{TP+E-MT}$  and  $PT_{TP+E-FT}$ , respectively. The results are very similar: the recall is 98.82% and 99.06%, the precision is 76.75% and 77.23%, and the  $F_1$ -score is 86.39% and 86.79% for  $F_{I_{RE-MT}}$  and  $F_{I_{RE-FT}}$ . In contrast, using  $PT_{TPE+MT}$  and  $PT_{TPE+FT}$  which were identified by considering exclusion patterns increases the precision with consistent recall for the extracted atomic software feature-relevant information: the precision of  $F_{I_{R-E+MT}}$  increased to 82.51% and the precision for

Table 5.5: Evaluation Results for Information Extraction

	$\mathbf{FI}_{\mathbf{RE}-\mathbf{MT}_G}$	$\mathbf{FI}_{\mathbf{RE}-\mathbf{FT}_G}$	$\mathbf{FI}_{\mathbf{R-E+MT}_G}$	$\mathbf{FI}_{\mathbf{R-E+FT}_G}$	$\mathbf{FI}_{\mathbf{RE+MT}_G}$	$\mathbf{FI}_{\mathbf{RE+FT}_G}$
TP	839	841	839	841	821	823
TN	300	301	375	378	494	495
FP	251	250	176	173	57	56
FN	10	8	10	8	28	27
R	98.82%	99.06%	98.82%	99.06%	96.69%	96.94%
P	76.75%	77.23%	82.51%	83.02%	93.37%	94.06%
$F_1$	86.39%	86.79%	89.93%	90.33%	95.00%	95.48%

$\mathbf{FI}_{\mathbf{R}[+|-]\mathbf{E}[+|-] [\mathbf{MT}|\mathbf{FT}]_G}$  = set of **Feature-relevant Information (FI)** extracted from software feature-relevant sentences (identified with **(E+)**/without **(E-)** **Exclusion patterns**) with **(R+)**/without **(R-)** considering syntactical **Relevancy** by using a **Full (FT<sub>G</sub>)** or a **Minimum Terminology Gold standard (FI<sub>G</sub>)**

$\mathbf{FI}_{\mathbf{R-E+FT}}$  increased to 83.02%. Thus, the answer to Q1.5 is that precision and  $F_1$  score can be increased by considering exclusion patterns during Information Identification. When considering syntactical relevancy during Information Extraction, the precision of the extracted atomic software feature-relevant information can be increased to 93.37% for  $\mathbf{FI}_{\mathbf{RE+MT}}$  and to 94.06% for  $\mathbf{FI}_{\mathbf{R-E+FT}}$ . The recall decreases slightly, because some units of information which are actually software feature-relevant are syntactically irrelevant and therefore are considered software feature-irrelevant. Thus, the answer to Q1.6 is that the consideration of syntactic relevancy in Information Extraction increases the precision with higher degree as it decreases recall. As a consequence, the  $F_1$ -score can be increased. According to Q1.7, the evaluations show that SOFEX works slightly better when using a full domain terminology instead of using a minimal domain terminology with regard to precision, recall, and  $F_1$ -score. Overall, SOFEX allows to extract atomic software feature-relevant information with an  $F_1$ -score of 95.00% ( $\mathbf{FI}_{\mathbf{RE+MT}}$ ) by means of  $\mathbf{MT}_G$  and 95.48% ( $\mathbf{FI}_{\mathbf{RE+FT}}$ ) by means of  $\mathbf{FT}_G$  which answers Q.1.

#### 5.2.4 Information Assignment (As)

Initially, Information Assignment was considered to cluster the atomic software feature-relevant units of information into software feature clusters in an unsupervised way. Therefore, the unsupervised clustering algorithm Hierarchical Agglomerative Clustering (HAC) was applied. HAC works bottom-up and initially treats each data instance as a single cluster. Iteratively, the clusters are merged based on some distance metrics (here Euclidean distance) until all clusters have been merged into a single cluster which contains all data instances (see, e.g., Aggarwal and Zhai, 2012a). In contrast to e.g.,

KM and EM, HAC does not require to know the number of clusters. The results show, that only 17% of the atomic software feature-relevant units of information are correctly clustered. Additionally, this approach would require to label the created clusters manually by domain experts. Thus, (1) the performance regarding classification quality is too low (see R.5 in Section 4.1) and would require additional manual correction effort and (2) the effort to label the clusters is too high (see R.4 in Section 4.1) in order to consider unsupervised clustering in SOFEX. Therefore, SOFEX uses classification algorithms in order to recommend software features for a corresponding atomic software feature-relevant information.

Information Assignment requires an extracted set of atomic software feature-relevant information ( $\mathbf{FI}_{\text{RE+MT}}$  and  $\mathbf{FI}_{\text{RE+FT}}$ ) as well as an existing feature hierarchy ( $\mathbf{FH}$ ) as input to assign each atomic software feature-relevant unit of information to its corresponding software feature. Evaluations showed that there is only a minor impact in using  $\mathbf{FI}_{\text{RE+MT}}$  or ( $\mathbf{FI}_{\text{RE+FT}}$ ) on the results of Information Assignment because the sets are very similar related to the atomic software feature-relevant information that is to be assigned to software features. This occurs because the number of TP which are to be assigned to software features differs only slightly between the two sets: 821 in  $\mathbf{FI}_{\text{RE+MT}}$  and 823 in  $\mathbf{FI}_{\text{RE+FT}}$  (see Table 5.5). Therefore, the following evaluation is based on  $\mathbf{FI}_{\text{RE+FT}}$  and aims to determine the best classification algorithm in order to recommend software features for a corresponding atomic software feature-relevant information. Furthermore, it is investigated to which degree preprocessing of the atomic software feature-relevant information impacts the results of Information Assignment. Similar to the evaluation of Information Identification (see Section 5.2.1), the two classification algorithms NB and SVM are chosen for consideration.

In context of Information Assignment, the following questions need to be answered:

Q1.8 Which classification algorithm provides best recommendation results?

Q1.9 To which degree does preprocessing affect recommendation results?

Table 5.6 shows an overview of the evaluation results related to Information Assignment. The header of the table refers to the different sets of atomic software feature-relevant information to software feature assignment as the result of Information Identification. The lines  $P_x$  list the precision for the different classification options:  $P_{aut}$  (i.e.  $P_1$ ) refers to a fully automated classification and  $P_{tN}$  refers to the precision of the recommended software feature within the top- $N$  classified software features. Thus, e.g.,  $P_{t5}$  indicates the precision that the correct software feature for a given atomic software feature-relevant unit of information is listed in the top-5 recommended software features based on the classification distribution. The lines  $t_x$  refer to time values in relation to the classification:



Table 5.6: Evaluation Results for Information Assignment

	<b>!AS<sub>PP+S</sub></b>	<b>AS<sub>PP+S</sub></b>	<b>AS<sub>PP-S</sub></b>	<b>!AS<sub>PP+N</sub></b>	<b>AS<sub>PP+N</sub></b>	<b>AS<sub>PP-N</sub></b>
$P_{aut}$	8.41%	8.77%	6.16%	13.86%	13.87%	7.07%
$P_{t3}$	8.41%	25.12%	18.13%	10.66%	19.79%	6.99%
$P_{t5}$	12.91%	47.16%	43.48%	21.68%	38.86%	19.79%
$P_{t10}$	28.20%	86.97%	77.01%	45.97%	69.08%	42.89%
$t_c$	15ms	15ms	15ms		1.5ms	
$t_u$	0ms	8200ms	8200ms	0ms	<1ms	<1ms
$t_i$		6100ms			28ms	

$AS_{C1[+][-][S][N]}$  = set of software feature-relevant information Assigned to software features with (Pp+)/without (Pp-) text preprocessing by using updated/not updated (!) Naive Bayes (N)/SVM (S) classification algorithm

1.  $t_c$ : time (in ms) required to classify a single atomic software feature-relevant unit of information
2.  $t_u$ : time (in ms) required to update a classifier with a new assignment of a single atomic software feature-relevant unit of information to its corresponding software feature
3.  $t_i$ : time (in ms) required to initially build the classifier with the 953 training instances extracted from the Feature Document

(**AS<sub>PP+S</sub>**, **AS<sub>PP-S</sub>**, **AS<sub>PP+N</sub>** and **AS<sub>PP-N</sub>**) (**!AS<sub>PP+S</sub>** and **!AS<sub>PP+N</sub>**).

The first observation is that updated classifiers, in general, perform better compared to classifiers which are not updated and only initially trained (indicated by the leading callsign “!” in Table 5.6). The second observation is that an updated SVM, in general, outperforms an updated NB in relation to software feature recommendations: SVM recommends software features with a precision of 25.12% for  $t_3$ , 47.16% for  $t_5$ , and 86.97% for  $t_{10}$  based on preprocessed text data (**AS<sub>PP+S</sub>**). In comparison, NB recommends software features with a precision of 19.79% for  $t_3$ , 38.86% for  $t_5$ , and 69.08% for  $t_{10}$  (**AS<sub>PP+N</sub>**). The tendency of the results is similar for a not-updated SVM (**!AS<sub>PP+S</sub>**) in comparison to a not-updated NB (**!AS<sub>PP+N</sub>**) classifier. In contrast, the performance of NB for a fully automated assignment ( $P_{aut}$ ) is higher compared to SVM, independent of the used setting: NB allows to automatically assign atomic software feature-relevant units of information to software features with precisions of 13.86% (**!AS<sub>PP+N</sub>**), 13.87% (**AS<sub>PP+N</sub>**), and 7.07% (**AS<sub>PP-N</sub>**). In contrast, the precisions of SVM are only between 6.16% (**AS<sub>PP-S</sub>**) and 8.77% (**AS<sub>PP+S</sub>**). Overall, the answer to Q1.8 is that SVM performs better for software feature recommendations while NB outperforms SVM if atomic software feature-relevant

units of information are fully automated assigning to software features.

The main drawback of SVM is depicted in the lower part of Table 5.6: if a SVM classifier needs to be updated, the time  $t_u$  required for updating is in average 8200ms per update and thus higher compared an average  $t_u$  of 28ms to update an updateable NB classifier. The values provided for  $t_u$  are average values: updating the SVM classifier with the first assignment between an atomic software feature-relevant unit of information and its corresponding software feature took about 4100ms. In contrast, the update of the SVM classifier with the last assignment of the 849 atomic software feature-relevant unit of information and its corresponding software feature took about 12300ms. This occurs because each update of the SVM classifier rebuilds the entire classification model with the training data. While the training data initially counts 952 records of assigned software feature-relevant units of information extracted from the Feature Document, the training data for the last update additionally contains the 849 records which were manually assigned by the domain expert. In contrast, the updateable NB classifier can be updated with a new training record in  $<1$ ms. Furthermore, the recommendation of a software feature for a single atomic software feature-relevant information lasts 15ms for SVM compared to only 1.5ms for the NB classifier. Summarized, the answer for Q1.9 is that a NB classifier performs faster compared to a SVM classifier related to classifying a single instance, updating the classifier, and initially training the classifier.

### 5.2.5 Discussion

The evaluation of Information Identification (see Section 5.2.1) shows that the exclusion patterns defined in context of SOFEX contribute to increase the recall of Information Identification. They allow to identify sentences which are software feature-irrelevant in context of the GDC user manual. Nevertheless, in other domains some pattern might not indicate software feature-irrelevant sentences: another domain expert might consider e.g., formulas, software feature-relevant. Thus, SOFEX allows to easily add or remove exclusion patterns to/from a configuration file. The domain terminology-based identification of software feature-relevant sentences outperforms approaches which use clustering and classification algorithms. The major problem of these approaches is the sparsity of the textual data (see, e.g., Phan et al., 2008; Rangrej et al., 2011): due to its shortness, the text does not provide enough co-occurrence, common context or a meaningful term frequency (mostly 1 per term in a sentence) in order to calculate appropriate similarity measures. Thus, it is very difficult to determine the degree of similarity between short texts.

The evaluation of IR Preprocessing (see Section 5.2.2) shows that the applied textual modifications and parse tree transformations allow to significantly increase the parse

tree accuracy. On the one hand, terminology-based textual modifications (see Section 4.4.1.1) can be used in other domains too without any customization: they indicate complex nouns for a POS parser and prevent incorrect POS tagging (see, e.g., Figure 4.9). On the other hand, pattern-based parse tree transformations (see Section 4.4.1.2) which are defined in context of the GDC user manual, might lead to undesired results in other domains: parse trees which are considered incorrect in the GDC context might be considered correct in another domain. Nevertheless, SOFEX allows to add, modify, and delete patterns and corresponding operations in/from a configuration file.

Information Extraction allows to extract atomic software feature-relevant information with an  $F_1$ -score of up to 95.48% (see Section 5.2.3) based on domain terminology gold standards. The evaluation shows that considering syntactical relevancy patterns significantly increases the precision of the extracted atomic software feature-relevant units of information ( $\sim +10\%$ ). But, syntactical relevancy also indicated true software feature-relevant information to be software feature-irrelevant information (FN, see Figure 5.2.3). Thus, the precision decreases slightly ( $\sim -2\%$ ). As an example, the atomic information “The correction factor can be switched off” consists of a subject (“The correction factor”) and a predicate (“can be switched off”) in context of the LIM model (see Section 2.3). Investigations showed that most units of information which correspond to the syntactic LIM-based pattern “subject + predicate”, are considered software feature-irrelevant. The reason is that a (at least the GDC) user manual contains a lot of system status notifications which follow that pattern (e.g., “The quantificator runs”) and are considered software feature-irrelevant. Nevertheless, the positive impact of the syntactical relevancy ( $\sim +10\%$  precision compared with  $\sim -2\%$  recall) predominates.

Information Assignment aims to recommend logically related software features for atomic software feature-relevant units of information. Evaluation show that updated classifiers outperform not updated classifiers (see Section 5.2.4) with an updated SVM classifier outperforming an updated NB classifier. The reason is that updated classifiers are provided a larger set of training data in form of continuously provided new data. SVM usually tends to perform better compared to NB because it considers the relations between the “textual features” which correspond to terms whereas NB treats them independently (see, e.g., Dong and Han, 2004; Hassan et al., 2011). Although an updateable SVM classifier performs better than an updateable NB classifier in context of Information Assignment, it is questionable if an average time to update the classification model (and thus the time to wait for a domain expert to classify another atomic software feature-relevant unit of information) of more than 8 seconds is acceptable in practice.

## 5.3 Evaluation II

The second evaluation aims to answer research question Q.2. Different to the the first evaluation (see Section 5.2), this evaluation reports the evaluation results using domain terminologies which are semi-automatically extracted by two domain experts from Roche: Domain Expert 1 ( $\mathbf{MT}_1$  and  $\mathbf{FT}_1$ ) and Domain Expert 2 ( $\mathbf{MT}_2$ ,  $\mathbf{FT}_2$ , and participant of the  $\mathbf{FT}_G$  creation). The reason to use domain terminologies extracted by two different domain experts is to (1) investigate the objectivity of domain terms (e.g., can the same domain terms objectively be identified from different domain experts?) and (2) to determine the impact of different domain terminologies on the results of the application of SOFEX.

Figure 5.3 shows a detailed overview of the different evaluations of SOFEX based on semi-automatically extracted full and minimal domain terminologies. The evaluations of Information Identification (see Section 5.3.2), IR Preprocessing (see Section 5.3.3), Information Extraction (see Section 5.3.4), and Information Assignment (see Section 5.3.5) use metrics similar to the corresponding evaluations of the first evaluation (see Section 5.2). Additionally, Terminology Extraction (see Section 4.3.4) is evaluated by means of precision, recall, and  $F_\beta$  (see Section 5.3.1). In contrast to all the other evaluations, a considerably higher value for  $\beta$  is used in order to determine the F-score because recall is considered more important than precision in context of Terminology Extraction: it is necessary to extract as many domain terms as possible to avoid overlooking software feature-relevant sentences during Information Identification (see, e.g., Berry et al. 2012). Finally, Section 5.3.6 discusses the results of the evaluation and primarily the impact of different domain terminologies on the results of the application of SOFEX.

### 5.3.1 Terminology Extraction (Tx)

Several fully automated approaches to extract domain terminologies exist in literature (see, e.g., Balachandran and Ranathunga, 2016; Kim et al., 2009; Venu et al., 2016), most of them originating from the area of Ontology Learning and aiming at extracting ontology concepts. Initially, the two approaches from Balachandran and Ranathunga (2016) and Venu et al. (2016) were considered for SOFEX because of their promising results, different strategies, and considerable effort to be implemented. But the evaluation showed that neither approach was sufficient enough to be applied within SOFEX and supported the decision to provide a semi-automated approach within SOFEX to extract domain terminologies:

- The approach of Balachandran and Ranathunga (2016) considers multiple domain corpora to extract domain terms from a specific domain. The idea is to weight each term regarding its relation to the given domain. This approach extracted

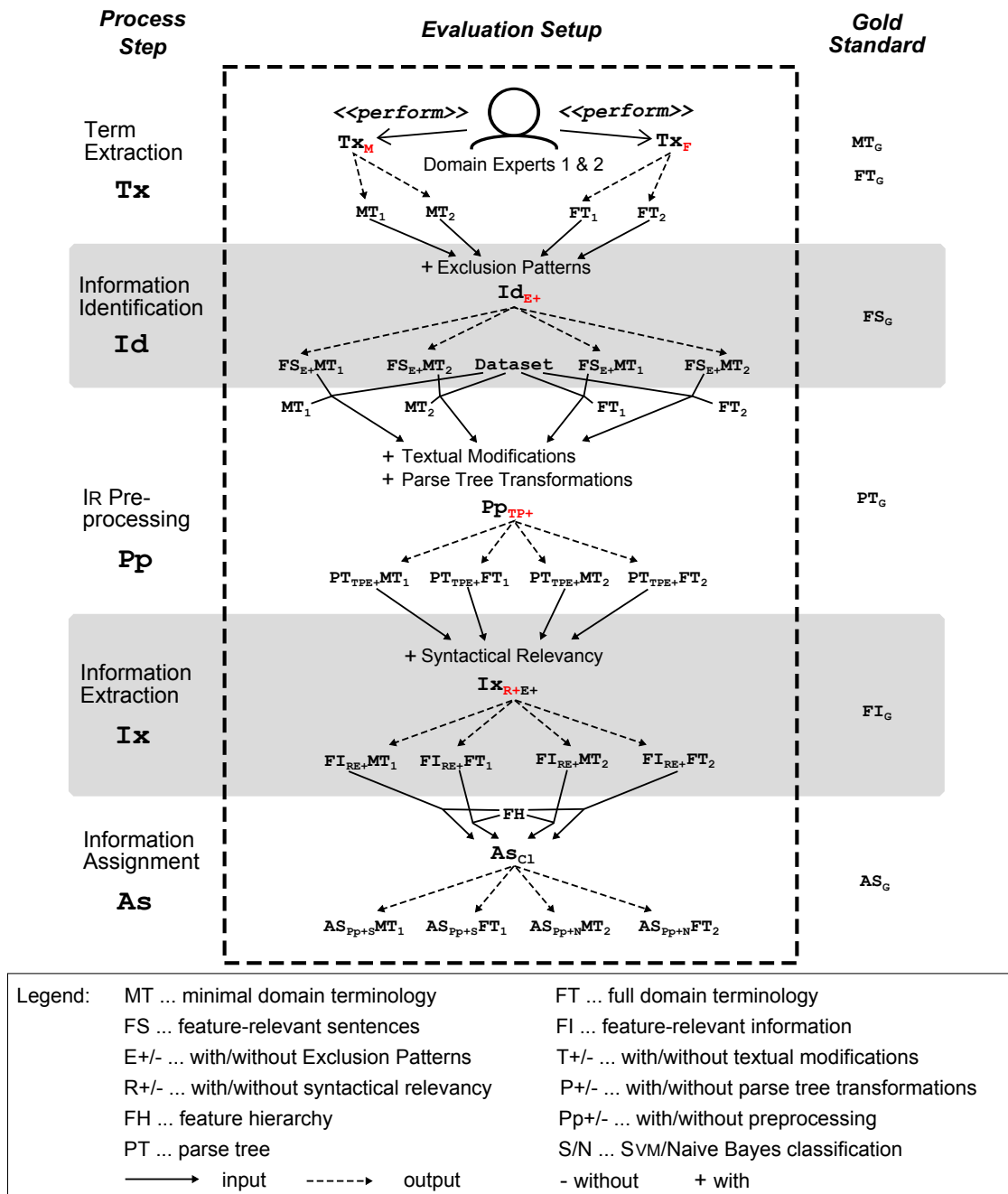


Figure 5.3: Evaluation Overview (Domain Experts)

542 domain terms in total with the highest precision in the top-40 terms of 19.3% (recall of 4.7%), and the highest recall of 34.1% (precision of 10.1%) in all 542 terms.

- The approach of Venu et al. (2016) aims to extract domain terms by means of the HITS algorithm (Kleinberg, 1999). The algorithm recursively calculates scores for

hubs expressed by semantic relations and authorities expressed as nouns. Finally, authorities that correspond to multi-grams and hubs that correspond to unigrams with the highest scores constitute the domain terms. The approach extracted 973 domain terms in total with the highest precision in the top-10 terms of 27.3% (recall of 1.8%), and the highest recall of 51.7% (precision of 3.2%) in all 973 terms.

As a first step, each domain expert performs Terminology Extraction twice to create a minimal and a full domain terminology:  $\text{MT}_1$  and  $\text{FT}_1$  are created by Domain Expert 1 whereas  $\text{MT}_2$  and  $\text{FT}_2$  are created by Domain Expert 2. In general, the objectives of the Terminology Extraction evaluation is to determine the objective identifiability of domain terms, to investigate commonalities and variabilities in comparison to the domain terminology gold standards, and to establish the effort for domain experts to extract domain terminologies by means of SOFEX. Thus, the following questions need to be answered:

- Q2.1 Are there differences in the number of terms between the different domain terminologies extracted by a specific domain expert (e.g.,  $\text{MT}_n$  vs.  $\text{FT}_n$ )?
- Q2.2 Are there differences in the number of terms between specific domain terminologies extracted by different domain experts (e.g.,  $\text{MT}_n$  vs.  $\text{MT}_m$ )?
- Q2.3 To which extent do the domain terminologies extracted by domain experts cover the domain terminology gold standards (e.g.,  $\text{MT}_G$  vs.  $\text{MT}_n$ )?
- Q2.4 Is the manual effort to create a minimal domain terminology much lower compared to the creation of a full domain terminology?

In total, the example sections of the user manual contain 1549 unique noun phrases. After step 1 and 2 of the candidate revision (see Section 4.3.4.1), the candidate list is reduced to 913 valid noun phrases (-41%). Afterwards, the *candidate split* is applied, resulting in a candidate list of 1059 noun phrases. Finally, the candidate list is sorted and then used as the basis for the semi-automatic term validation (see Section 4.3.4.2).

Table 5.7 shows an overview of the characteristics and metrics of the different domain terminologies used. The two gold standards  $\text{MT}_G$  and  $\text{FT}_G$  contain 80 and 119 terms, respectively. The table depicts that the minimal domain terminologies contain a much lower number of domain terms compared to the corresponding full domain terminologies which allows to answer Q2.1:  $\text{MT}_1$  comprises 56.78% fewer terms than  $\text{FT}_1$ ,  $\text{MT}_2$  comprises 27.54% fewer terms than  $\text{FT}_2$ , and  $\text{MT}_G$  comprises 37.77% fewer terms than  $\text{FT}_G$ . Furthermore, the answer to Q2.2 is that the domain terminologies differ strongly with regard to the number of domain terms among the different domain experts: while Domain Expert 1 determined 118 and 271 domain terms for  $\text{MT}_1$  and  $\text{FT}_1$ , Domain Expert 2 determined

273 and 374 domain terms for  $\mathbf{MT}_2$  and  $\mathbf{FT}_2$ , respectively. Thus, there exist differences in the number of terms between domain terminologies extracted by different domain terminologies. In order to answer Q2.3, the Jaccard coefficient (see, e.g., Chen et al. 2004) is used to compare the similarity of sets. It is defined as the size of the intersection (e.g.,  $\#\text{terms}_{\mathbf{MT}_G} \cap \#\text{terms}_{\mathbf{MT}_1}$ ) divided by the size of the union (e.g.,  $\#\text{terms}_{\mathbf{MT}_G} \cup \#\text{terms}_{\mathbf{MT}_1}$ ) of the sets (e.g.,  $\mathbf{MT}_1 = (80-26)/(118+26) = 0.375$ ) Overall, the similarity between the domain terminologies created by the domain experts and the gold standards is very low, between 18% and 38%.

The lower part of Table 5.7 shows the standard metrics. The number of true positives TP corresponds to the number of terms in the semi-automatically extracted domain terminologies  $\mathbf{MT}_1$ ,  $\mathbf{MT}_2$ ,  $\mathbf{FT}_1$ , and  $\mathbf{FT}_2$  which are contained in the corresponding domain terminology gold standards  $\mathbf{MT}_G$  and  $\mathbf{FT}_G$ . The number true negatives TN corresponds to the number of term candidates which were truly determined to be not specific for the GDC domain by the domain experts. The precision for the semi-automatically extracted domain terminologies ranges from 18.11% for  $\mathbf{MT}_2$  up to 45.76% for  $\mathbf{MT}_1$ . In contrast, the recall ranges from 60.00% for  $\mathbf{MT}_2$  and 80.67% for  $\mathbf{FT}_2$ . Last, the table depicts three F-scores with different  $\beta$  values in order to be able to draw conclusions regarding the importance of precision and recall of domain terminologies in the subsequent process steps. In general, the F-score increases with increased  $\beta$  because recall is throughout higher than precision.

Table 5.7: Evaluation Results for Domain Terminology Extraction

	$\mathbf{MT}_G$	$\mathbf{MT}_1$	$\mathbf{MT}_2$	$\mathbf{FT}_G$	$\mathbf{FT}_1$	$\mathbf{FT}_2$	
# terms	80	118	271	119	273	374	
# missing (FN)		26	32		31	23	
# additional (FP)		64	217		185	278	
size diff. [%]		+47.50	+238.75		+129.41	+214.29	
similarity [%]		37.50	18.18		28.95	24.18	
# TP*		54	48		88	96	
# TN**		915	762		755	662	
Precision [%]		45.76	18.11		32.23	25.67	
Recall [%]		67.50	60.00		73.95	80.67	
F <sub>0.5</sub> [%]		48.91	21.05		36.33	29.72	
F <sub>1</sub> [%]		54.54	27.82		44.89	38.95	
F <sub>10</sub> [%]		67.18	58.66		73.01	78.99	

\* TP =  $\#\text{terms}[\mathbf{M}|\mathbf{F}]\mathbf{T}_G - \text{FN}$   
\*\* TN =  $1059 - \text{TP} - \text{FN} - \text{FP}$   
 $[\mathbf{M}|\mathbf{F}]\mathbf{T}_{[G|1|2]} = \mathbf{Full}(\mathbf{F}r)$  or a **Minimum (M<sub>r</sub>) Terminology** produced by domain expert **1** or domain expert **2** in comparison to the **Gold standards** (grey shade).

In order to extract a full domain terminology, a domain expert has to validate each of the 1059 single candidate term. In contrast, when creating a minimal domain terminology, Domain Expert 1 validated only 822 candidate terms (-22%) . Thus, the answer to Q2.4

is that the effort to semi-automatically extract a minimal domain terminology is less compared to extracting a full domain terminology. This occurs, because several candidate terms which contain a validated domain term are not considered for further validation (see Section 4.3.4.2). Nevertheless, each domain terminology is further used for Information Identification in order to determine the impact of different domain terminologies on the ability of SOFEX to identify software feature-relevant sentences.

### 5.3.2 Information Identification (Id)

Information Identification uses the sentences of the *Dataset* and a domain terminology as input to determine which sentences are potentially software feature-relevant (e.g.,  $\mathbf{FS}_{E+MT_1}$ ) or which are not. In contrast to Information Identification with  $\mathbf{MT}_G$  and  $\mathbf{FT}_G$  (see Section 5.2), Information Identification is only applied considering exclusion patterns ( $\mathbf{ID}_{E+}$ ). This is done as it was already shown that the application of exclusion patterns increases the ability to identify potential software feature-relevant sentences correctly. Furthermore, the results of Information Identification with semi-automatically extracted domain terminologies are compared to the results of Information Identification using  $\mathbf{FT}_G$  because it provided the best results (see Section 5.2.1). In that context, the focus of our research questions moves to the usage of the domain terminologies extracted by the domain experts:

Q2.5 Do semi-automatically extracted domain terminologies impact the recall, precision, and  $F_1$ -score of Information Identification?

Table 5.8 depicts the results of Information Identification with semi-automatically extracted domain terminologies ( $\mathbf{FS}_{E+}[\mathbf{FT}|\mathbf{MT}]_{[1|2]}$ ) compared to Information Identification with  $\mathbf{FT}_G$  ( $\mathbf{FS}_{E+}\mathbf{FT}_G$ , see Section 5.2). By means of semi-automatically extracted domain terminologies, Information Identification achieves a recall of at least 91.79% for  $\mathbf{FS}_{E+}\mathbf{FT}_2$  up to 97.47% for  $\mathbf{FS}_{E+}\mathbf{MT}_1$  compared to a recall of 98.57% for  $\mathbf{FS}_{E+}\mathbf{FT}_G$ . The precision for Information Identification with semi-automatically extracted domain terminologies is between 78.54% for  $\mathbf{FS}_{E+}\mathbf{FT}_1$  and 80.16% for  $\mathbf{FS}_{E+}\mathbf{MT}_2$  compared to 80.69% for  $\mathbf{FS}_{E+}\mathbf{FT}_G$ . Similarly, the  $F_1$ -score for Information Identification with semi-automatically extracted domain terminologies is between 85.44% for  $\mathbf{FS}_{E+}\mathbf{FT}_2$  and 87.52% for  $\mathbf{FS}_{E+}\mathbf{MT}_1$  compared to 88.81% for  $\mathbf{FS}_{E+}\mathbf{FT}_G$ . Thus, the evaluation results allow to answer Q2.5: there is indeed a dependency between the set of identified potentially software feature-relevant sentences and the used domain terminology. Nevertheless, the results differ only slightly. Compared to the metrics of the different domain terminology, Information Identification tends to perform better the higher the precision of the domain terminology.

Table 5.8: Evaluation Results for Information Identification



	$\mathbf{FS}_{\mathbf{E}+\mathbf{FT}_G}$	$\mathbf{FS}_{\mathbf{E}+\mathbf{MT}_1}$	$\mathbf{FS}_{\mathbf{E}+\mathbf{MT}_2}$	$\mathbf{FS}_{\mathbf{E}+\mathbf{FT}_1}$	$\mathbf{FS}_{\mathbf{E}+\mathbf{FT}_2}$
TP	625	616	597	613	579
TN	371	361	374	353	375
FP	150	160	147	168	146
FN	6	15	34	18	52
R	98.57%	97.47%	94.62%	97.16%	91.79%
P	80.69%	79.41%	80.16%	78.54%	79.92%
F <sub>1</sub>	88.81%	87.52%	86.79%	86.86%	85.44%

$\mathbf{FS}_{\mathbf{E}+[\mathbf{MT}|\mathbf{FT}]_{[G|1|2]}}$  = set of **F**eature-relevant **S**entences (**FS**) which was determined by considering **E**xclusion patterns (**E+**) and using a **F**ull (**FT**) or a **M**inimum (**MT**) **T**erminology produced by domain expert **1** or domain expert **2** in comparison to the **G**old standard (grey shade).

### 5.3.3 IR Preprocessing (Pp)

The domain terminologies from Domain Expert 1 ( $[\mathbf{MT}|\mathbf{FT}]_1$ ) and Domain Expert 2 ( $[\mathbf{MT}|\mathbf{FT}]_2$ ) as well as the sets of identified potentially software feature-relevant sentences ( $\mathbf{FS}_{\mathbf{E}+[\mathbf{FT}|\mathbf{MT}]_{[1|2]}}$ ) are used as input for IR Preprocessing. In contrast to IR Preprocessing with domain terminology gold standards (see Section 5.2.2), IR Preprocessing is applied considering textual modifications and parse tree transformations as it was already shown that their application increases the parse tree accuracy. In that context, the research question aims to determine the impact of semi-automatically extracted domain terminologies on IR Preprocessing:

Q2.6 Do semi-automatically extracted domain terminologies impact IR preprocessing?

Table 5.9: Parse Tree Accuracy for IR Preprocessing

$\mathbf{PT}_{\mathbf{TPE}+\mathbf{MT}_G}$	$\mathbf{PT}_{\mathbf{TPE}+\mathbf{MT}_1}$	$\mathbf{PT}_{\mathbf{TPE}+\mathbf{MT}_2}$	$\mathbf{PT}_{\mathbf{TPE}+\mathbf{FT}_G}$	$\mathbf{PT}_{\mathbf{TPE}+\mathbf{FT}_1}$	$\mathbf{PT}_{\mathbf{TPE}+\mathbf{FT}_2}$
94.81%	94.12%	93.73%	98.43%	97.91%	96.32%

$\mathbf{PT}_{\mathbf{TPE}+[\mathbf{MT}|\mathbf{FT}]_{[G|1|2]}}$  = set of **P**arse **T**rees (**PT**) generated from software feature-relevant sentences (identified by considering **E**xclusion patterns - **E+**) with **T**extual modification and **P**arse tree transformations (**TP+**) by using a **F**ull (**FT**) or a **M**inimum **T**erminology (**MT**) produced by domain expert 1 or domain expert 2 in comparison to the **G**old standard (grey shade).

In context of minimal domain terminologies, Table 5.9 depicts that the parse tree accuracy is at least 93.73% for  $\mathbf{PT}_{\mathbf{TPE}+\mathbf{MT}_2}$  compared to 94.81% for  $\mathbf{PT}_{\mathbf{TPE}+\mathbf{MT}_G}$ . By using full domain terminologies, the parse tree accuracy is at least 96.32% for  $\mathbf{PT}_{\mathbf{TPE}+\mathbf{FT}_2}$  compared to 98.43% for  $\mathbf{PT}_{\mathbf{TPE}+\mathbf{FT}_G}$ . Thus, the answer to Q2.6 is that semi-automatically extracted domain terminologies affect the parse tree accuracy as a result of IR Preprocessing to only a minor degree. In contrast to Information Identification, IR Preprocessing tends to deliver more accurate results the higher the recall of the used domain terminology.

### 5.3.4 Information Extraction (Ix)

Information extraction requires the identified potential software feature-relevant sentences as input to determine the set of atomic software feature-relevant units of information (e.g.,  $\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_1}$ ). Similar to information identification and IR preprocessing, the evaluation of information extraction is only performed with the optimal setting ( $\mathbf{IX}_{\mathbf{R}+\mathbf{E}+}$ , see Section 5.2).

In that context, the research question aims to determine the impact of semi-automatically extracted domain terminologies on information extraction:

Q2.7 Do semi-automatically extracted domain terminologies impact IR preprocessing?

Table 5.10: Evaluation Results for Information Extraction

	$\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_G}$	$\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_1}$	$\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_2}$	$\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_G}$	$\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_1}$	$\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_2}$
TP	821	805	792	823	808	771
TN	494	494	488	495	493	495
FP	57	57	63	56	58	56
FN	28	44	57	26	41	78
R	96.69%	94.71%	93.15%	96.94%	95.05%	90.80%
P	93.37%	93.28%	92.61%	94.06%	93.19%	93.22%
F <sub>1</sub>	95.00%	93.99%	92.88%	95.48%	94.11%	92.00%

$\mathbf{FI}_{\mathbf{RE}+[\mathbf{MT}|\mathbf{FT}]_{[G|1|2]}}$  = set of **Feature-relevant Information (FI)** extracted from software feature-relevant sentences (identified with **Exclusion patterns - E+**) considering syntactical **Relevancy (R+)** by using a **Full (Fr)** or a **Minimum Terminology (Mt)** produced by domain expert 1 or domain expert 2 in comparison to the **Gold standard** (grey shade).

Table 5.10 presents the evaluation results for Information Extraction using sets of potentially software feature-relevant sentences identified by using minimal domain terminologies ( $\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_{[1|2]}}$ ) and full domain terminologies ( $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_{[1|2]}}$ ). Using minimal domain terminologies, the extracted sets of atomic software feature-relevant information have a recall, precision, and a F<sub>1</sub>-score at least 93.15%, 92.61%, and 92.88% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_2}$  compared to 96.69%, 93.37%, and 95.00% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{MT}_G}$ . The metrics for Information Extraction using full domain terminologies are similar: recall is at least 90.80% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_2}$  compared to 96.94% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_G}$ ; precision is at least 93.19% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_1}$  compared to 94.06% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_G}$ ; F<sub>1</sub>-score is at least 92.00% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_2}$  compared to 95.48% for  $\mathbf{FI}_{\mathbf{RE}+\mathbf{FT}_G}$ . To answer Q2.7, the usage of semi-automatically extracted domain terminologies impacts Information Extraction compared to using domain terminology gold standards. Overall and similar to Information Identification, the results for Information Extraction in relation to F<sub>1</sub>-score tend to be slightly better the higher the precision of the extracted domain terminology.

### 5.3.5 Information Assignment (As)

Information Assignment requires an extracted set of atomic software feature-relevant information ( $\mathbf{F}_{\text{IRE+MT}}$  and  $\mathbf{F}_{\text{IRE+FT}}$ ) as well as an existing Feature Hierarchy ( $\mathbf{FH}$ ) as input to assign each atomic software feature-relevant unit of information to its corresponding software feature. The results of the evaluation in Section 5.2.4 showed that the best results for Information Assignment are achieved with (1) updated classifiers and (2) text preprocessing before classification. Thus, similar to the evaluation of Information Identification, IR Preprocessing, and Information Extraction, the evaluation of Information Assignment is only performed with the optimal setting ( $\mathbf{AS}_{\text{CL+S}}$  and  $\mathbf{AS}_{\text{CL+N}}$ , see Section 5.2). In context of this evaluation, the research question aims to determine the impact of semi-automatically extracted domain terminologies on Information Assignment:

Q2.8 Do semi-automatically extracted domain terminologies indirectly impact Information Assignment?

Table 5.11: Evaluation Results for Information Assignment

(a) Information Assignment with Naive Bayes

	$\mathbf{AS}_{\text{PP+N}}$	$\mathbf{AS}_{\text{PP+NMT}_1}$	$\mathbf{AS}_{\text{PP+NMT}_2}$	$\mathbf{AS}_{\text{PP+NFT}_1}$	$\mathbf{AS}_{\text{PP+NFT}_2}$
$P_{\text{aut}}$	13.87%	12.95%	12.85%	12.94%	12.93%
$P_{t3}$	19.79%	18.48%	18.33%	18.46%	18.45%
$P_{t5}$	38.86%	36.29%	36.00%	36.26%	36.23%
$P_{t10}$	69.08%	64.51%	63.99%	64.45%	64.40%

(b) Information Assignment with SVM

	$\mathbf{AS}_{\text{PP+S}}$	$\mathbf{AS}_{\text{PP+SMT}_1}$	$\mathbf{AS}_{\text{PP+SMT}_2}$	$\mathbf{AS}_{\text{PP+SFT}_1}$	$\mathbf{AS}_{\text{PP+SFT}_2}$
$P_{\text{aut}}$	8.77%	8.19%	8.12%	8.18%	8.18%
$P_{t3}$	25.12%	23.46%	23.27%	23.44%	22.42%
$P_{t5}$	47.16%	44.04%	43.69%	44.00%	43.97%
$P_{t10}$	86.97%	81.22%	80.56%	81.15%	81.08%

Table 5.11 shows an overview of the different evaluation results related to Information Assignment with NB and SVM classifiers. The grey shaded columns refers to average values of Information Assignment of based on minimal and full domain terminologies with the optimal setting, taken from Section 5.2.4 for comparison purpose. Similar to the evaluation of Information Assignment using the atomic software feature-relevant gold standards (see Section 5.2.4), the evaluation provides the precision metrics for fully automated classification ( $P_{\text{aut}}$ ) as well as software feature recommendation (top- $N$ ,  $P_{tN}$ ).

Basically, the results for Information Assignment using atomic software feature-relevant information extracted by means of the domain terminologies from Domain Expert 1 and Domain Expert 2 are consistently worse using NB (see Table 5.11(a)) as well as SVM (see Table 5.11(b)) compared to atomic software feature-relevant information extracted by means of the domain terminology gold standards ( $\mathbf{AS}_{\text{CL+S}}$  and  $\mathbf{AS}_{\text{CL+S}}$ , grey shaded). The main reason behind this finding is that each of the false positive atomic feature-relevant information (see Section 5.3.4) is wrongly classified or recommended. Answering Q2.8, using semi-automatically extracted domain terminologies does (indirectly) impact Information Assignment.

### 5.3.6 Discussion

The evaluation of the domain terminologies (see Section 5.3.1) shows that the semi-automatically extracted domain terminologies differ substantially regarding the number of domain terms, missing and additional domain terms, and furthermore the domain term coverage in comparison to the domain terminology gold standards. In particular, the domain terminologies extracted by Domain Expert 2 differ strongly regarding the number of captured domain terms in comparison to the corresponding domain terminology gold standards:  $\mathbf{MT}_2$  and  $\mathbf{FT}_2$  contain by 238.75% and 214.29% more domain terms with a coverage of 18.18% and 24.18%, respectively. In contrast, the domain terminologies from Domain Expert 1, namely  $\mathbf{MT}_1$  and  $\mathbf{FT}_1$ , contain “only” 47.50% and 129.41% more domain terms than the corresponding domain terminology gold standards. This indicates that an objective determination of domain terms is difficult in general: even domain experts from the same domain might interpret the concept of a domain term differently. Furthermore, it is actually difficult to reproduce an extracted domain terminology several times (e.g., by performing Terminology Extraction): Domain Expert 1 already participated in the creation of the domain terminology gold standard  $\mathbf{FT}_G$  about two years before applying the Terminology Extraction from SOFEX. Still, his extracted domain terminologies ( $\mathbf{MT}_1$  and  $\mathbf{FT}_1$ ) differ from the domain terminology gold standards.

Table 5.8 in Section 5.3.2 shows, that domain terminologies impact the identification of potentially software feature-relevant sentences: depending on the terms captured in a domain terminology, in particular the recall is affected (e.g.,  $\mathbf{FS}_{E+}$   $\mathbf{FT}_{E+}$ ). This is because the number of additional domain terms in the corresponding domain terminology lead to an incorrect identification of potentially software feature-relevant sentences. As an example,  $\mathbf{FT}_2$  contains 374 terms in total and 278 additional terms in comparison to the gold standard (see  $\mathbf{FT}_G$  in Table 5.7 of Section 5.3.1). This leads to 52 sentences which were wrongly identified to be potentially software feature-relevant compared to only 6 by using  $\mathbf{FT}_G$  (see FN in Figure 5.8). This conclusion is additionally supported when considering that

Information Identification performs slightly better by using minimal domain terminologies than using full domain terminologies with average  $F_1$ -scores of 87.16% and 86.15%, respectively. Nevertheless, Information Identification is not analogously impacted in relation to the strongly differing domain terminologies. The reason behind is that the domain experts were able to determine the domain terms which frequently occur in software feature-relevant sentences.

Similarly, the usage of different domain terminologies also impacts IR Preprocessing. Table 5.9 in Section 5.3.3 shows that IR Preprocessing using full domain terminologies outperforms IR Preprocessing using minimal domain terminologies based on the average  $F_1$ -scores: using full domain terminologies yield a parse tree accuracy of 97.12% compared to 93.93% by means of minimal domain terminologies. This occurs because of the interpretation of domain terms by the POS parser: each n-gram domain term is treated as a single word instead of n separate words which may result in different parse trees as output of IR Preprocessing. As an example, the POS parser treats the phrase “material list” differently depending on the domain terminology. On the one hand, a full domain terminology  $\mathbf{FT}_x$  contains the domain term *material list*. By applying the textual modifications (see Section 4.4.1) the phrase is modified to “*DTmaterialDTlist*” and furthermore correctly tagged as single noun by the POS parser. On the other hand, a minimal domain terminology ( $\mathbf{MT}_x$ ) does not contain *material list* but *material*. Thus, the the phrase “material list” is modified to “*DTmaterial list*”. The POS parser then identifies “*DTmaterial*” as single noun and “*list*” wrongly as a verb which distorts the entire parse tree. Thus, the more n-gram domain terms are captured in a corresponding domain terminology, the more parse trees of potentially software feature-relevant sentences are created correctly by the POS parser.

Finally, using a minimal or a full domain terminology impacts the extraction of atomic software feature-relevant information only slightly. Table 5.10 in Section 5.3.4 shows that both, using a minimal as well as a full domain terminology, leads to similar results. The average  $F_1$ -scores are 93.44% using  $\mathbf{MT}$  and 93.06% using  $\mathbf{FT}$ . To answer the question, which domain terminology needs to be created, therefore depends on two considerations. First, the intended use of the domain terminology needs to be considered: a full domain terminology contains a complete overview of domain terms and can thus be used for e.g., stakeholder communication. In comparison, a minimal domain terminology is usually not complete (see Section 4.3.4). Second, the effort to extract a domain terminology needs to be considered: the extraction of a minimal domain terminology is considerably faster compared to the extraction of a full domain terminology (see Section 4.3.4). In order to only extract atomic software feature-relevant information, a minimal domain terminology is sufficient.

## Part V Conclusion

# Chapter 6

## Discussion

This chapter discusses the effort needed to apply SOFEX per se as well as the effort needed to adapt SOFEX for possible other contexts and domains (see Section 6.1). Section 6.2 discusses the threats to validity.

### 6.1 Application Effort & Adaption Need

Basically, SOFEX was designed with a focus on minimal manual effort (see requirement R.4). Nevertheless, some process steps require human intervention to apply SOFEX appropriately. Furthermore, SOFEX was designed in context of Roche's GDC user manual. However, the design of SOFEX allows to customize several process steps and characteristics in order to apply it in other contexts and domains. Thus, the following two sections discuss the manual effort needed (1) to apply SOFEX on the GDC user manual and (2) to customize SOFEX in order to apply it to other domains and contexts. The discussions provide an overview and estimation of the manual effort required to apply SOFEX.

#### 6.1.1 Manual Effort to Apply SOFEX

Based on experiences, the amount of manual work for the process steps *User Manual Revision* (see Section 4.3.1), *Terminology Extraction* (see Section 4.3.4), and *Information Assignment* (see Section 4.5) can be estimated. In total, the revision of the user manual excerpts took approximately one hour (6 minutes per page in average). Nevertheless, the time required strongly depends on the quality of the corresponding user manual to be revised, the reviewer's abilities, as well as the number of figures contained. Validating the domain terminology took between 30 minutes for 822 domain term candidates ( $\mathbf{MT}_1$ ) and 60 minutes for 1059 domain term candidates ( $\mathbf{FT}_2$ ). Last, Information Assignment is the

most costly task regarding manual effort. The creation of the software feature-information gold standard which is the assignment of atomic software feature-relevant information to the corresponding software features took approximately 5 hours (20 seconds per atomic software feature-relevant unit of information in average) for 849 atomic software feature-relevant units of information. In comparison, using the recommendations from SOFEX, the effort decreased by approximately 70%, down to 1.5 hours which corresponds to 6.5 seconds per atomic software feature-relevant unit of information using top-10 software feature suggestions. This is, because for more than 80% of the atomic software feature-relevant units of information, the correct logically related software feature is listed in the top-10 suggested software features.

Table 6.1: Manual Effort to Apply SOFEX (in hours)

Pages	User Manual Revision	Terminology Extraction	Information Assignment	Total
50	~0.5	~0.5 (~850 candidates)	~1.5 (~850 units of inf.)	~2.5 hours
500	~5	~1.5 (~2.750 candidates)	~15 (~8.500 units of inf.)	~21.5 hours

Table 6.1 depicts an overview of the effort required to apply SOFEX: for 50 pages, the effort is 2.5 hours in order to enhance existing software features with corresponding atomic software feature-relevant information. In order to extract atomic software feature-relevant information from a user manual with 500 pages and enhance existing software features with that information, the effort would increase to approximately 21.5 hours: the effort for User Manual Revision and Information Assignment is increased tenfold. However, the larger the training data (e.g., assigned atomic software feature-relevant information) for the classification model of Information Assignment is, the more accurate the recommended software features become. Hence, the average time to decide on a software feature to assign the information might decrease. As already mentioned, in case the user manual is syntactically correct, the entire effort for it's revision is eliminated (see Section 4.3.1). Investigations showed that there will not be a linear increasing amount of new domain term candidates in new user manual sections. In context of the GDC user manual, 10% of the domain term candidates which were extracted from the second exemplary section were already extracted from the first exemplary section and are thus duplicates. Originating from approximately 20 domain term candidates per user manual page <sup>1</sup> and a decrease of 10% of new domain term candidates (no duplicates) per 25 pages (e.g., for the pages 26-50 -10% of new domain term candidates based on 500 domain terms from the first section = 450 new domain terms) <sup>2</sup>, the estimated number of domain

<sup>1</sup> each of the two analysed user manual sections comprises about 25 pages and contained approximately 500 domain term candidates resulting in 20 domain term candidates per page

<sup>2</sup> for the pages 51-75 -20% = 400 new domain terms, for the pages 76-100 -30% = 350 new domain



term candidates is about 2750 for a 500 pages user manual.

### 6.1.2 Manual Effort to Adapt SOFEX

Document Preparation (see Section 4.3.2) aims to extract NL text and sentence types from a user manual and provide an internal data representation. The sentence types might differ from user manual to user manual, but can easily be adapted in a configuration file of SOFEX. The sentence types represent the style sheets used in a corresponding Word document(see Figure 4.4 in Section 4.3.2). Thus, they can be easily determined resulting in low effort in exchanging them.

SOFEX identifies potentially software feature-relevant sentences by means of domain terms, their sentence types, and exclusion patterns (see Section 4.3). The sentence types are used to determine software feature-irrelevant sentences (see 5. in Section 4.3.5) and correspond to style sheets within a Word document. The time required to identify the relevant sentence types can only be estimated. Usually, there are only some style sheets used in a document (e.g., headings, bullets, standard). Additionally, there exist a few style sheets from the used style sheets (e.g., heading) which will most likely not contain any software feature-relevant information (3 out of 8 in case of the GDC user manual, see 5. in Section 4.3.5) and thus refer to software feature-irrelevant sentences. These style sheets may be verified at first glance (e.g., by investigating the table of contents). Other style sheets which do not indicate software feature-irrelevant sentences at first glance must be investigated in detail. Nevertheless, based on experiences, the uncovering of style sheets which indicate software feature-irrelevant sentences can be made with little effort.

Similarly, the effort required to determine phrases and sentences which can be considered software feature-irrelevant (e.g., within brackets, containing the word “section”, 1.-4. in Section 4.3.5) can be estimated. These lexical-based exclusion patterns are defined via regular expressions in the configuration file of SOFEX. Basically, the exclusion patterns are obvious because they usually appear frequently (e.g., references to sections and figures, phrases enclosed in parentheses) and are thus easily determined. Nevertheless, adding exclusion patterns is only worth as long as it significantly reduces the amount of sentences to be further analyzed. The effort to determine and add exclusion patterns can be considered low: in context of the GDC user manual, all exclusion patterns were identified after reading the first three pages of the first exemplary section.

IR Preprocessing (see Section 4.4.1) aims to provide parse trees for each potentially software feature-relevant sentence by means of a POS parser. Furthermore, automated textual and parse tree modifications are applied to increase the accuracy of the parse

---

terms, ...

trees. Textual-based parse tree modifications (see Section 4.4.1.1) is a fully automated task and only based on the domain terms which are semi-automatically extracted. Thus, there is no additional effort which needs to be considered. Similarly, the application of pattern-based parse tree transformations (see Section 4.4.1.2) is fully automated. But, in contrast, the definitions of patterns to identify incorrect (parts of) parse trees and corresponding modifications (see the Appendices B.1 and B.2 for details and examples) require a significantly higher effort:

1. a user needs to identify incorrect parse trees in the set of all parse trees from the potentially software feature-relevant sentences
2. a user needs to define the pattern (Tregex pattern) which matches the incorrect (part of) a parse tree
3. a user needs to define corresponding manipulation operations (Tsurgeon operations) in order to correct an incorrect parse tree or only parts of it.

Syntactical relevancy patterns (see 4. in Section 4.4.2) finally determine if a potentially software feature-relevant atomic unit of information is truly software feature-relevant. The user has to identify these syntactical patterns and add them to the SOFEX configuration file. Similarly to parse tree transformations, the user needs to investigate all parse trees. Nevertheless, the identification of the patterns can be considered less expensive because

- there are fewer patterns to be defined
- there is no need to define transformation operations
- the patterns are based on the LIM-syntax and are thus not as complex as Tregex patterns which are defined on POS-tag-level.

## 6.2 Threats to Validity

This section discusses the four main threats to validity according to Wohlin et al. (2012) of the evaluation results of SOFEX and the measures taken to minimize these threats. Validity in context of research investigates if the conclusions drawn from an observation might be wrong (e.g., relationship between reality and conclusion Feldt and Magazinius, 2010).

### 6.2.1 Threats to Conclusion Validity

Conclusion validity refers to the possibility to draw correct conclusions regarding the relationship between the treatments and the outcome of an experiment. In context of

this thesis, the statistical measures used (precision, recall, and  $F_\beta$ -score) are mature and proven. Thus, they provide reliable results.

### 6.2.2 Threats to Internal Validity

Internal validity refers to the extent to which a procedure influenced the result. User Manual Revision was applied to two different sections of the user manual by two different people. The sections were revised by two different non domain experts who got instructed a priori. For evaluation purpose, several different gold standards were used ( $\mathbf{MT}_G$ ,  $\mathbf{FT}_G$ ,  $\mathbf{FS}_G$ ,  $\mathbf{FI}_G$ ,  $\mathbf{AS}_G$ , see Section 5.2) which were created by domain experts. Those domain experts were not involved in the development and implementation of SoFEX. Thus, the gold standards are not influenced by any development tasks. One domain expert (Domain Expert 2, see Section 5.3) participated in the creation of the full terminology gold standard  $\mathbf{FT}_G$  two years before applying Terminology Extraction. Nevertheless, the results in Table 5.7 (see Section 5.3.1) show that  $\mathbf{FT}_G$  strongly differs from the semi-automatically extracted domain terminology  $\mathbf{FT}_2$ . Thus, Domain Expert 2 is considered being unbiased by participating in the creation of  $\mathbf{FT}_G$ .

### 6.2.3 Threats to Construct Validity

Construct validity refers to the degree to which a treatment does what it claims to do. I have not been involved in the development of the source (GDC user manual) or the gold standards. Thus, the view of what constitutes a software feature, software feature-relevant sentence or information is a clearly external one. In contrast, the gold standards have been created by domain experts. It is assumed that the domain experts do clearly know which sentence or information is considered software feature-relevant and which not. As there does not exist a clear objective definition of a software feature or software feature-relevant information in theory and practice (see Section 2.3), even the domain experts often did not agree on the software feature-relevancy of specific artifacts (sentence, information). Thus, the results with other domain experts might differ.

### 6.2.4 Threats to External Validity

External validity refers to the extent to which an approach can be applied in other contexts. In the following, the generalizability of SoFEX and furthermore its applicability in another domain is discussed. Basically, the quality of the extracted atomic software feature-relevant units of information and furthermore the software feature description relies on the following five key aspects:

1. the quality of the user manual to be analysed

2. the quality of the extracted domain terminology
3. the accuracy of the parse trees of the corresponding software feature-relevant sentences from the user manual
4. the syntactical relevancy patterns
5. the proper assignment of the atomic software feature-relevant units of information to the corresponding software features.

The first and second aspects cannot be influenced by SOFEX. They rather rely on the abilities of the responsible editors and are thus the same for each domain. The accuracy of the parse trees (3rd aspect) relies on the terminology-based textual modifications and the pattern-based parse tree transformations (besides the used POS parser API). The quality of the terminology-based textual modifications depend on the extracted domain terminology. Thus, they are domain independent and rather depend on the individual extracting the domain terminology and his/her ability to “correctly” determine domain terms. Pattern-based parse tree transformations, on the other hand, modify parse trees which are considered “incorrect”. These parse trees are at least considered “incorrect” in the GDC domain. But, in order to determine the “correctness” of parse trees in other domains, SOFEX needs to be applied in that domains. Apart from the identification and definition of the patterns and corresponding transformation operations, the patterns and operations can easily be maintained in the configuration file of SOFEX. The syntactical relevancy patterns (fourth aspect) are used by SOFEX in order to finally determine true atomic software feature-relevant units of information based on their LIM syntax. Similar to the accuracy of parse trees, the relevancy patterns which are defined in context of GDC may not be sufficient to determine all atomic software feature-relevant units of information in other domains.

SOFEX has not been applied in other domains or a different context due to time restrictions from available domain experts. Hence, SOFEX can be overfitted to the GDC context as the application of parse tree modifications and syntactical relevancy patterns will not deliver similar results in other domains. Nevertheless, while designing and implementing SOFEX, attention was paid to preferably use general patterns to be reusable in other contexts.

## Summary

This thesis contributes to the body of knowledge in SE and RE with respect to the analysis of and methods for the extraction of software feature-relevant information from NL user manuals. Chapter 3 investigates state of the art approaches which allow to extract software feature-relevant information from NL software engineering artifacts. The SLR focuses on the NL software engineering artifacts mined, the extracted information, and the extracted type of software feature-relevant information. Based on these findings and the requirements expressed by Roche, SOFEX is designed in Chapter 4. Each component of SOFEX is finally evaluated in Chapter 5. In total, SOFEX provides four main contributions:

1. Semi-automated Domain Terminology Extraction: SOFEX allows to automatically extract domain term candidates from a NL user manual. Furthermore, it provides automated support for domain experts in order to validate true domain terms from the candidates (see Section 4.3.4) which is evaluated in an empirical study (see Section 5.3.1) and compared to fully automated approaches.
2. Automated Software Feature-relevant Information Identification: SOFEX allows to automatically identify sentences in a NL user manual which potentially contains software feature-relevant information by means of a domain terminology and structural information (see Section 4.3) with high precision and recall (see Sections 5.2.1 and 5.3.2).
3. Automated Extraction of Atomic Software Feature-relevant Information: SOFEX allows to automatically extract atomic software feature-relevant units of information from potentially software feature-relevant sentences by means of syntactic patterns (see Section 4.4) with high precision and recall (see Sections 5.2.3 and 5.3.4).
4. Recommendation-based Software Feature Knowledge Enhancement: Finally,

SoFEX supports the assignment of atomic software feature-relevant information to corresponding, logically related, software features by means of a text classification approach. It recommends the top-N software features for an atomic software feature-relevant information based on the likelihood of their logical relationship (see Section 4.5). This approach is evaluated in the Sections 5.2.4 and 5.3.5.

Overall, the evaluations show that SoFEX allows to identify and extract software feature-relevant information with high precision and recall in a more or semi-automatic manner. However, unifying, compressing, classifying, and clustering of short NL text data will likely be hard to process fully automated with promising results in a near future. Basically, the results of SoFEX highly depend on the parse tree accuracy and thus the syntax of the NL text provided in the corresponding user manual. SoFEX was developed in cooperation with Roche and is based on the GDC user manual. The development is based on investigations of two user manual sections. In order to validate SoFEX, several gold standards were used. The observations and experiences gained from the evaluation based on the two user manual sections show that SoFEX can work well for an entire user manual. However, the effort for the domain experts to create gold standards to apply SoFEX to the entire GDC user manual and evaluate the results was too high. The effort to apply SoFEX with similar results to other user manuals is difficult to judge without knowing the specifics. Nevertheless, this thesis presents an approach for software feature-relevant information identification and extraction and elaborates several empirical insights on software feature-relevant information for future research.

## Future Work

In general, SOFEX aims to support the retrospective extraction of software-feature relevant information and the enhancement of software feature knowledge. It was evaluated based on two exemplary sections of Roche’s GDC user manual by means of gold standards that were manually crafted. In order to prove the generalizability and applicability of SOFEX, it has to be adapted and applied to the entire GDC user manual, in other domains, and other contexts besides GDC.

The extracted software feature-relevant units of information are atomic. Therefore, their semantic context should be more clear in comparison to non-atomic information: each unit of information is as simple as possible and should therefore be clearly related to a single software feature only. As an example, the non-atomic software feature-relevant information “the material can be added to the material list and it can be deleted from the material list” might be related to two different software features (e.g., “extend material list” and “reduce material list”). SOFEX extracts the two atomic units of information (1) “the material can be added to the material list” and (2) “it can be deleted from the material list” from the exemplary sentence. The two units of information are then distinctively assigned to a single software feature (e.g., (1) is related to “extend material list” and (2) is related to “reduce material list”). Nevertheless, pronouns (personal, possessive, relative, reflexive) are still an issue within SOFEX. The second information (2) obviously shows that the pronoun fades the actual information (because the domain term “material” is represented by the personal pronoun “it”) which may complicate the assignment of that information to the logically related software feature (without knowing the first (1) information). Thus, considering anaphora resolution approaches which aim to resolve pronouns (see, e.g., Mitkov, 2014) in context of SOFEX may increase its ability to recommend correct software features for a corresponding software feature-relevant information to the domain expert.

Overall, the results of this thesis are promising and support the area of domain-specific information extraction (especially software feature-related information extraction). SOFEX shows that it is possible to identify domain-relevant information by means of a domain terminology and structural information with high precision and recall.



# Part VI Appendix

# Natural Language Processing

This chapter provides insights into NLP-related specifics which are used in context of SoFEX.

## **A.1 Penn Tag Set**

Table A.1 shows the PoS-Tags of the Penn Tag Set which were used in the context of SoFEX.

## **A.2 Stopwords**

Table A.2 depicts the stopwords which are used in context of SoFEX.

Table A.1: Penn Tag Set

Abb.	Description	Abb.	Description
ADJP	Adjective phrase	CC	Coordinating conjunction
ADVP	Adverb phrase	CD	Cardinal number
FRAG	Fragment	DT	Determiner
NP	Noun phrase	IN	Preposition or subordinating conjunction
PP	Prepositional phrase	JJ	Adjective
QP	Quantifier phrase	MD	Modal
VP	Verb phrase	NN	Noun, singular or mass
WHADJP	Wh-adjective phrase	NNS	Noun, plural
WHADVP	Wh-adverb phrase	NNP	Proper noun, singular
WHNP	Wh-noun phrase	NNPS	Proper noun, plural
WHPP	Wh-prepositional phrase	POS	Possessive ending
		PRP	Personal pronoun
		PRP\$	Possessive pronoun
		RB	Adverb
		SYM	Symbol
		TO	to
		VB	Verb, base form
		VBD	Verb, past tense
		VBG	Verb, gerund or present participle
		VBN	Verb, past participle
		VBP	Verb, non-3rd person singular present
		VBZ	Verb, 3rd person singular present
		WDT	Wh-determiner
		WP	Wh-pronoun

Table A.2: Stopword List

a	being	except	ie	nothing	someone	under
able	below	few	if	now	something	until
about	beside	fifteen	in	nowhere	sometime	up
above	besides	fify	inc	of	sometimes	upon
across	between	fill	indeed	off	somewhere	us
after	beyond	find	interest	often	step	use
afterwards	bill	fire	into	on	still	usually
again	both	first	is	once	such	very
against	bottom	five	it	one	take	via
all	but	following	its	only	ten	want
almost	by	for	keep	onto	than	was
alone	call	former	last	or	that	we
along	can	formerly	latter	other	the	well
already	cannot	forty	latterly	others	their	were
also	cant	found	least	otherwise	them	what
although	can't	four	less	our	themselves	whatever
always	co	from	ltd	ours	then	when
am	computer	front	made	ourselves	thence	whence
among	con	full	main	out	there	whenever
amongst	could	further	many	over	thereafter	where
amongst	couldnt	general	may	own	thereby	whereafter
amount	couldn't	generally	me	part	therefore	whereas
an	cry	get	meanwhile	per	therein	whereby
and	de	give	might	perhaps	thereupon	wherein
another	describe	go	mill	please	these	whereupon
any	detail	had	mine	put	they	wherever
anyhow	do	has	more	rather	thick	whether
anyone	done	hasnt	moreover	re	thin	which
anything	down	hasn't	most	same	third	while
anyway	due	have	mostly	see	this	whither
anywhere	during	he	move	seem	those	who
are	each	hence	much	seemed	though	whoever
around	eg	her	must	seeming	three	whole
as	eight	here	my	seems	through	whom
at	either	hereafter	name	serious	throughout	whose
back	eieven	hereby	namely	several	thru	why
basically	else	herein	neither	she	thus	will
be	elsewhere	hereupon	never	should	to	with
became	empty	hers	nevertheless	show	together	within
because	enough	herself	next	side	too	without
become	etc	him	nine	since	top	would
becomes	even	himself	no	sincere	toward	yet
becoming	ever	his	nobody	six	towards	you
been	every	how	none	sixty	twelve	your
before	everyone	however	noone	so	twenty	yours
beforehand	everything	hundred	nor	some	two	yourself
behind	everywhere	i	not	somehow	un	yourselves

# Appendix **B**

## Parse Tree Modification patterns

This chapter presents the patterns used in SOFEX which are required to (1) correct parse trees as well as (2) adapt parse trees in order to extract potentially feature-relevant information in a smooth way. The patterns are defined by means of Tregex (Levy and Andrew 2006) which indicate parts of a parse tree to be modified. Tsurgeon (Levy and Andrew 2006), which is a tree-transformation utility built on top of Tregex, allows to manipulate the identified parse trees as desired. In the following sections, we provide Tregex patterns with corresponding Tsurgeon operations and examples. An example shows a parse tree before modification on the left hand side, indicating the part of the parse tree which matches the pattern defined and are colored **red**. The right hand side shows the parse tree after modification(s) which are colored **green**.

### B.1 Parse tree correction patterns

#### B.1.1 JJ to NN

Pattern: `_____ < (!/NN.?/=rel < /^QD.*|^DST.*/)`

Operation: `[relabel rel NN]`

Example:

<code>(ROOT</code>	<code>(ROOT</code>
<code>  (ADVP (<b>JJ</b> DSTmanualDSTentry)))</code>	<code>  (ADVP (<b>NN</b> DSTmanualDSTentry)))</code>

#### B.1.2 ADVP to NP

Pattern: `_____=rel < (_____ < /DST|QD/ !> NP)`

Operation: `[relabel rel NP]`

Example:

(ROOT	(ROOT
(ADVP (NNS DSTmanualDSTentry)))	(NP (NNS DSTmanualDSTentry)))

### B.1.3 Cleanse PP

Pattern: VP=par <+(NP) (NP \$+ (/,/ <1 /,/ \$+ (\_\_\_\_\_ < (/,/ <1 /,/ \$+ PP=mov)))

Operation: [move mov >-1 par]

Example:

(ROOT	(ROOT
(S	(S
(NP (JJ other) (NNS options))	(NP (JJ other) (NNS options))
(VP (VBP exclude)	(VP (VBP exclude)
(NP	(NP
(NP (NNP DSTreagents))	(NP (NNP DSTreagents))
(, ,)	(, ,)
(SBAR	(SBAR
(WHNP (WDT which))	(WHNP (WDT which))
(S	(S
(VP (VBP are)	(VP (VBP are)
(VP (VBN used)	(VP (VBN used)
(PP (IN for)	(PP (IN for)
(NP	(NP
(NP (NNP DSTcontrols))	(NP (NNP DSTcontrols))
(CC and)	(CC and)
(NP (NNP DSTcalibs)))	(NP (NNP DSTcalibs)))
(, ,)	(, ,)))))
(PP (IN from)	(PP (IN from)
(NP (NN invoicing))))))	(NP (NN invoicing)))
(. .))	(. .))

### B.1.4 VP to JJ

Pattern: ROOT|SINV < (/ADVP|VP/=exc < \_\_\_\_\_=mov !< VBP)  
 <-1 (NP <+(NP) (/NN.?! > \_\_\_\_\_=par))

Operation: [relabel mov JJ][move mov >1 par][excise exc exc]

Example:

(ROOT	(ROOT
(SINV	(SINV
(VP (VBD entered))	(NP (JJ entered) (NNP DSTdealDSTdata))
(NP	(VP (VBG regarding))))
(NP (NNP DSTdealDSTdata))	
(VP (VBG regarding))))	

### B.1.5 ADJP to PP

Pattern: VP=par <+(NP|S) (NP \$+ (ADJP=rel < (VBN \$+ PP)))

Operation: [move rel >-1 par][relabel rel PP]

Example:

<pre>(ROOT  (S   (NP (DT the) (NNP DSTcustomer))   (VP (VBZ handles)     (S      (NP (DT the) (NNPS DSTmaterials))      (ADJP (VBN related)       (PP (TO to)         (NP (DT the) (NN offer))))))))))</pre>	<pre>(ROOT  (S   (NP (DT the) (NNP DSTcustomer))   (VP (VBZ handles)     (S      (NP (DT the) (NNPS DSTmaterials))      (PP (VBN related)       (PP (TO to)         (NP (DT the) (NN offer))))))))))</pre>
--	--

### B.1.6 Complex NP#1

Pattern: PP < NP=bro \$+ (NP=exc <1 NP)

Operation: [move exc \$- bro][excise exc exc]

Example:

<pre>(ROOT  (PP (IN on)   (NP (NNP DSTtest)))  (NP   (NP (NN level))   (PP (IN for)     (NP (NNP DSTreagents))))))</pre>	<pre>(ROOT  (PP (IN on)   (NP (NNP DSTtest))   (NP (NN level))   (PP (IN for)     (NP (NNP DSTreagents))))))</pre>
--	--

### B.1.7 Complex NP#2

Pattern: \_\_\_\_\_ < (NP=par < /NN.?[CD/ \$+ (NP=exc < /NN.?[CD/ < \_\_\_\_\_=mov !< CC))

Operation: [move exc \$- bro][excise exc exc]

Example:

<pre>(ROOT  (PP (IN on)   (NP (NNP DSTtest))   (NP (NN level))   (PP (IN for)     (NP (NNP DSTreagents))))))</pre>	<pre>(ROOT  (PP (IN on)   (NP (NNP DSTtest) (NN level))   (PP (IN for)     (NP (NNP DSTreagents))))))</pre>
--	---

### B.1.8 Complex NP#3

Pattern: NP <: NP=exc

Operation: [excise exc exc]

Example (cont.):

<pre>(ROOT  (S   (NP (NN DSTpack)        (NP (NN optimization) (NNS data))        (PP (IN for)             (NP              (NP (NNP DSTreagents))              (VP (VBG regarding))))))</pre>	<pre>(ROOT  (S   (NP (NN DSTpack) (NN optimization) (NNS data))   (PP (IN for)        (NP         (NP (NNP DSTreagents))         (VP (VBG regarding))))))</pre>
--	---

### B.1.9 Complex NP#4

Pattern: NP=par \$+ (NP=del <+(NP) (NP < \_\_\_\_\_=mov \$+ \_\_\_\_\_=mov2))

Operation: [move mov >-1 par][move mov2 \$- par][delete del]

Example:

<pre>(ROOT  (NP   (NP (DT the) (NN DSTpack))   (NP    (NP (NNS sizes))    (PP (IN of)         (NP (DT each) (NNP DSTmaterial))))))</pre>	<pre>(ROOT  (NP   (NP (DT the) (NN DSTpack) (NNS sizes))   (PP (IN of)        (NP (DT each) (NNP DSTmaterial))))))</pre>
--	--

### B.1.10 Complex NP#5

Pattern: /NP\$/=mov !< /NN.?[NP\$|PRP|EX/ \$+ NP=par

Operation: [move mov >1 par][excise mov mov]

Example:

<pre>(ROOT  (NP   (NP (JJ annual))   (NP (NNP DSTinstrument))))</pre>	<pre>(ROOT  (NP   (NP (JJ annual) (NNP DSTinstrument))))</pre>
---	--



### B.1.11 Cleanse PP

Pattern: /<sup>^</sup>NP\$|<sup>^</sup>PPN\$/=sis < (PP=mov !\$- /<sup>^</sup>NP\$|<sup>^</sup>PPN\$/)

Operation: [move mov \$- sis]

Example: (NP (NNP DSTrounding) was already modified from (VP (VBG DSTrounding) by applying *Cleanse NN#1*, *Cleanse NN#2*, and *Complex NP#2*

<pre>(ROOT   (S     (NP (NNP DSTrounding) (NNS effects)       (PP (IN of)         (NP (NNP DSTbatch))))))</pre>	<pre>(ROOT   (S     (NP (NNP DSTrounding) (NNS effects))     (PP (IN of)       (NP (NNP DSTbatch))))))</pre>
---	--

### B.1.12 Cleanse NP lists#1

Pattern: (NP <-1 /NN.\*/ >+(NP) (PP \$- NP=par))

Operation: if count(*par*) > 1 and  $par_i = par_j$ , then move  $par_j$  \$+  $par_i$

Example: *Cleanse NN#1*, *Cleanse NN#2*, and *Complex VP#1* are already applied

<pre>(ROOT   (S     (NP (DT the) (NN DSTannualDSTquantity))     (VP (VBZ is) (VBN calculated)       (PP (VBN based) (IN on)         (NP           (NP (DT the) (NN quantity))           (PP (IN of)             (NP               (NP                 (NP (NNP DDSTevents))                 (PP (IN per)                   (NP (NN year))))               (, ,)               (NP                 (NP (DT the) (NN quantity))                 (PP (IN of)                   (NP (NNP DSTsteps))))               (, ,)               (CC and)               (NP                 (NP (DT the) (NN quantity))                 (PP (IN of)                   (NP (NNP DSTtests))))))))))</pre>	<pre>(ROOT   (S     (NP (DT the) (NN DSTannualDSTquantity))     (VP (VBZ is) (VBN calculated)       (PP (VBN based) (IN on)         (NP           (NP (DT the) (NN quantity))           (PP (IN of)             (NP               (NP (NNP DDSTevents))               (PP (IN per)                 (NP (NN year))))               (, ,)               (NP                 (NP (DT the) (NN quantity))                 (PP (IN of)                   (NP (NNP DSTsteps))))               (, ,)               (CC and)               (NP                 (NP (DT the) (NN quantity))                 (PP (IN of)                   (NP (NNP DSTtests))))))))))</pre>
--	--

### B.1.13 Cleanse NP lists#2

Pattern: NP \$+ PP \$- \_\_\_\_\_=sis > (!/NP|VP/=nam > \_\_\_\_\_=par)

Operation: cover the NP and it's corresponding PP node with a NP node

Example: *Complex VP#1* and *Complex VP#1* are already applied

<pre>(ROOT   (PP (IN on)     (NP (NNP DSTtest) (NN level))     (PP (IN for)       (NP (NNP DSTreagents))))))</pre>	<pre>(ROOT   (PP (IN on)     (NP       (NP (NNP DSTtest) (NN level))       (PP (IN for)         (NP (NNP DSTreagents))))))</pre>
--	--

### B.1.14 Cleanse S#1

Pattern: \_\_\_\_\_ < (NP=son1 \$+ (VP=son2 \$+ /CC|,/))

Operation: [insert S=par \$+ son1][move son1 >-1 par][move son2 >-1 par]

Example: continuation from *Cleanse S#2*

<pre>(ROOT   (S     (SBAR (IN once)       (S         (NP (DT the) (NNPS QDstatus))         (VP (VBZ is) (VBN selected))))     (, ,)     (NP (DT the) (NNP DSTscenario))     (VP (VBZ is) (VBN locked))     (CC and)     (S       (NP (NNS changes))       (VP (VBP are) (RB not) (JJ possible)))     (. .)))</pre>	<pre>(ROOT   (S     (SBAR (IN once)       (S         (NP (DT this) (NNPS QDstatus))         (VP (VBZ is) (VBN selected))))     (, ,)     (S       (NP (DT the) (NNP DSTscenario))       (VP (VBZ is) (VBN locked)))     (CC and)     (S       (NP (NNS changes))       (VP (VBP are) (RB not) (JJ possible)))     (. .)))</pre>
--	---

### B.1.15 Cleanse S#2

Pattern: NP=mov1 [\$+ VP=mov2 | \$- VP=mov2] [\$++ S | \$- S] > (S >+(S) ROOT)

Operation: [insert S=par \$+ mov1][move mov1 >-1 par][move mov2 >-1 par]

Example: *Complex VP#1* is already applied

```

(ROOT
(S
(S
(NP (DT each) (NNP DSTscenario))
(VP (MD can) (VB be) (VBN assigned)
  (PP (TO to)
    (NP (DT a) (NNP DSTcustomer))))))
(, ,)
(NP (DT each) (NNP DSTcustomer))
(VP (MD can) (VB be) (VBN assigned)
  (PP (TO to)
    (NP (NNS DSTgroups))))))
(. .))

(ROOT
(S
(S
(NP (DT each) (NNP DSTscenario))
(VP (MD can) (VB be) (VBN assigned)
  (PP (TO to)
    (NP (DT a) (NNP DSTcustomer))))))
(, ,)
(S
(NP (DT each) (NNP DSTcustomer))
(VP (MD can) (VB be) (VBN assigned)
  (PP (TO to)
    (NP (NNS DSTgroups))))))
(. .))

```

### B.1.16 Cleanse "between" #1

Pattern: PP=par < (IN < between) < (\_\_\_\_\_ <1 /.P/ !< /CC|,/ « (/CC|,/=mov1 \$+ \_\_\_\_\_=mov2))

Operation: [move mov1 >-1 par][move mov2 >-1 par]

Example:

```
(ROOT
(S
(NP (EX there))
(VP (VBZ exists)
(NP
(NP (DT a) (NN difference))
(PP (IN between)
(NP
(NP (DT a) (NN DSTtest))
(VP (VBN displayed)
(PP (IN in)
(NP
(NP (DT the) (NN DSTlist))
(CC and)
(NP
(NP (DT a) (NN DSTtest))
(VP (VBN used)
(PP (IN in)
(NP (DT the) (NN DSTarea))))))
(. .)))
```

```
(ROOT
(S
(NP (EX there))
(VP (VBZ exists)
(NP
(NP (DT a) (NN difference))
(PP (IN between)
(NP
(NP (DT a) (NN DSTtest))
(VP (VBN displayed)
(PP (IN in)
(NP
(NP (DT the) (NN DSTlist))))))
(CC and)
(NP
(NP (DT a) (NN DSTtest))
(VP (VBN used)
(PP (IN in)
(NP (DT the) (NN DSTarea))))))
(. .)))
```

### B.1.17 Cleanse "between" #2

Pattern: PP=par < (IN < between) !< CC !< (NP < CC) >+(NP) (NP \$+ (CC=mov1 \$+ NP=mov2))

Operation: [move mov1 >-1 par][move mov2 >-1 par]

Example:

```
(ROOT
(S
(NP (DT the) (NN status) (NN QDadd))
(VP (VBZ indicates)
(SBAR (IN that)
(S
(NP
(NP
(NP (DT the) (NN assignment))
(PP (IN between)
(NP (NNP DSTproductDSTfamily))))
(CC and)
(NP
(NP (VBG DSTrounding) (NN mode))
(PP (IN in)
(NP (DT the) (NN database))))))))))
(. .)))
```

```
(ROOT
(S
(NP (DT the) (NN status) (NN QDadd))
(VP (VBZ indicates)
(SBAR (IN that)
(S
(NP
(NP
(NP (DT the) (NN assignment))
(PP (IN between)
(NP (NNP DSTproductDSTfamily))
(CC and)
(NP
(NP (VBG DSTrounding) (NN mode))
(PP (IN in)
(NP (DT the) (NN database))))))))))
(. .)))
```

## B.2 Parse tree adaption patterns

### B.2.1 Remove SINV

Pattern: SINV=exc

Operation: [excise exc exc]

Example:

<pre>(ROOT   (SINV     (NP       (NP (JJ used) (NNP DSTmaterial))))))</pre>	<pre>(ROOT   (NP     (NP (JJ used) (NNP DSTmterial))))))</pre>
---	--

### B.2.2 Remove Brackets

Pattern: PRN=del <1 -LRB-

Operation: [delete del]

Example:

<pre>(ROOT   (NP     (NP (DT the) (NNP DSTmaterial))     (PRN (-LRB- -LRB-))     (NP (NNP e.g.))     (, ,)     (NP (NNP DSTreagent))     (-RRB- -RRB-))))</pre>	<pre>(ROOT   (NP     (NP (DT the) (NN DSTmaterial))))</pre>
---	---

### B.2.3 Cleanse FRAG

Pattern: \_\_\_\_\_=nam > FRAG=rel

Operation: replace each FRAG with *nam*

Example:

<pre>(ROOT   (FRAG     (NP (NN calculation))     (PP (IN without)       (NP (NNP DSTcooledDSTstability))))))</pre>	<pre>(ROOT   (NP     (NP (NN calculation))     (PP (IN without)       (NP (NNP DSTcooledDSTstability))))))</pre>
--	--

### B.2.4 Complex VP#1

Pattern: VP <- (VP=exc !\$ VP)

Operation: [excise exc exc]

Example:

```
(ROOT
  (S
    (NP (DT the) (NNP DSTquantificator))
    (VP (VBZ starts)
      (S
        (VP (TO to)
          (VP (VB run))))))
    (. .)))
```

```
(ROOT
  (S
    (NP (DT the) (NNP DSTquantificator))
    (VP (VBZ starts)
      (S
        (VP (TO to) (VB run))))))
  (. .)))
```

### B.2.5 Complex VP#2

Pattern: VP=par < /MD|VB.?!JJ/=sis > (VP < /MD|VB.?!JJ/=del)

Operation: add *sis* to each corresponding VB (in a loop) and delete *del* afterwards

Example:

```
(ROOT
  (S
    (NP (DT the) (NN user))
    (VP (MD can)
      (VP
        (VP (VB remove)
          (NP (NNS materials))
          (PP (IN from)
            (NP (DT the) (NNP DSTscenario))))
          (, ,)
          (CC or)
          (VP (VB modify)
            (NP (PRP$ their) (NN quantity))))))
    (. .)))
```

```
(ROOT
  (S
    (NP (DT the) (NN user))
    (VP
      (VP (MD can) (VB remove)
        (NP (NNS materials))
        (PP (IN from)
          (NP (DT the) (NNP DSTscenario))))
      (, ,)
      (CC or)
      (VP (MD can) (VB modify)
        (NP (PRP$ their) (NN quantity))))
    (. .)))
```

### B.2.6 Complex VP#3

Pattern: VP < (/VB.?! \$ . ADJP|ADVP=exc)

Operation: [excise exc exc]

Example:

<pre>(ROOT  (S   (NP    (NP (DT the) (NN default) (NN name))    (PP (IN of)     (NP (DT a) (NNP DSTsystemDSTgroup))))   (VP (VBZ is)    (ADJP (JJ configurable)     (PP (IN in)      (NP (DT the) (NNP DSTadminDSTtool))))   (. .)))</pre>	<pre>(ROOT  (S   (NP    (NP (DT the) (NN default) (NN name))    (PP (IN of)     (NP (DT a) (NNP DSTsystemDSTgroup))))   (VP (VBZ is) (JJ configurable)    (PP (IN in)     (NP (DT the) (NNP DSTadminDSTtool))))   (. .)))</pre>
--	---

### B.2.7 Complex VP#4

Pattern: VP < (/VB.\*/ \$. ADJP|ADVP=exc)

Operation: [excise exc exc]

Example:

<pre>(ROOT  (S   (NP (NNP DSTgdc))   (VP (VBZ creates)    (NP (DT a) (NNP DSTsystemDSTgroup))    (ADVP (RB automatically)))   (. .)))</pre>	<pre>(ROOT  (S   (NP (NNP DSTgdc))   (VP (RB automatically) (VBZ creates)    (NP (DT a) (NNP DSTsystemDSTgroup)))   (. .)))</pre>
---	---

### B.2.8 ADVP in VP#1

Pattern: ADVP=exc \$+ /VB.\*/

Operation: [excise exc exc]

Example:

<pre>(ROOT  (S   (NP (DT the) (NN user))   (VP (MD can)    (ADVP (RB manually))    (VP (VB adapt)     (NP (DT the) (NNS DSTsettings))))   (. .)))</pre>	<pre>(ROOT  (S   (NP (DT the) (NN user))   (VP (MD can) (RB manually))    (VP (VB adapt)     (NP (DT the) (NNS DSTsettings))))   (. .)))</pre>
---	--

### B.2.9 ADVP in VP#2



Pattern: ADVP=exc < \_\_\_\_\_=mov \$+ VP=par

Operation: [move mov >1 par][excise exc exc]

Example:

```
(ROOT
  (S
    (NP (DT the) (NN user))
    (ADVP (RB manually))
    (VP (VBZ adapts)
      (NP (DT the) (NN DSTfrequency)))
    (. .)))
```

```
(ROOT
  (S
    (NP (DT the) (NN user))
    (VP (RB manually) (VBZ adapts)
      (NP (DT the) (NN DSTfrequency)))
    (. .)))
```

### B.2.10 ADJP in VP

Pattern: NP|VP <+(S) ADJP=exc

Operation: [excise exc exc]

Example:

```
(ROOT
  (S
    (NP (DT the) (NNS reagents))
    (VP (VBP are)
      (ADJP (JJ available)))
    (. .)))
```

```
(ROOT
  (S
    (NP (DT the) (NNS reagents))
    (VP (VBP are) (JJ available))
    (. .)))
```

### B.2.11 PRT in VP

Pattern: VP < PRT=exc

Operation: [excise exc exc]

Example:

```
(ROOT
  (S
    (NP (DT the) (NN sum))
    (VP (VBZ is)
      (VP (VBN rounded)
        (PRT (RP up))))
    (. .)))
```

```
(ROOT
  (S
    (NP (DT the) (NN sum))
    (VP (VBZ is)
      (VP (VBN rounded) (RP up)))
    (. .)))
```

### B.2.12 Complex PP

Pattern: NP=bro < (IN=mov \$.. \_\_\_\_\_=mov2)

Operation: [insert (PP=par) \$- bro][move mov >-1 par]  
 [insert (NP=par2) >-1 par][move mov2 >-1 par2]

Example: considering the application of *Cleanse ADJP* beforehand (striked through)

<pre>(ROOT  (S   (VP (VB change)    (NP     (NP (NNP DSTenvironmentDSTdata))     <del>(ADJP (IN if) (JJ necessary)))</del>    (. )))</pre>	<pre>(ROOT  (S   (VP (VB change)    (NP     (NP (NNP DSTenvironmentDSTdata))     (PP (IN if)      (NP (JJ necessary))))    (. )))</pre>
--	---

### B.2.13 Complex NP#6

Pattern: \_\_\_\_\_ < (NP=par \$+ /NN.?/=mov)

Operation: [move mov >-1 par]

Example:

<pre>(ROOT  (NP   (NP (DT the) (NNP DSTmaterial) (POS 's))   (NN quantity)))</pre>	<pre>(ROOT  (NP   (NP (DT the) (NNP DSTmaterial) (POS 's) (NN quantity))))</pre>
--	--

### B.2.14 Multiple PP#1

Pattern: \_\_\_\_\_=sis < (\_\_\_\_\_ < (PP <: IN=bro) \$+ (CC \$+ (\_\_\_\_\_ < (PP <-1 \_\_\_\_\_=ins \$+ \_\_\_\_\_=mov))))

Operation: [move mov >-1 par]

Example:

```

(ROOT
  (S
    (VP
      (VP (VB remove)
        (NP (NNS DSTmaterials))
        (PP (IN from)))
      (CC or)
      (VP (VB change)
        (NP (JJ DSTmaterial) (NNS quantities))
        (PP (IN in)
          (NP (DT a) (NNP DSTdeal)))))))))

```

```

(ROOT
  (S
    (VP
      (VP (VB remove)
        (NP (NNS DSTmaterials))
        (PP (IN from)
          (NP (DT a) (NNP DSTdeal))))))
      (CC or)
      (VP (VB change)
        (NP (JJ DSTmaterial) (NNS quantities))
        (PP (IN in)
          (NP (DT a) (NNP DSTdeal)))))))))

```

### B.2.15 Multiple PP#2

Pattern: VP < (VP=par !< NP \$+ /,|CC/) <-1 (VP < (PP=ins1 \$- (NP=ins2 !>+(NP) PP !\$- PP)))

Operation: [insert ins1 >-1 par][insert ins2 >-2 par]

Example:

```

(ROOT
  (S
    (NP (NNP DSTgdc))
    (VP (VBZ provides)
      (S
        (NP (NNS possibilities))
        (VP
          (VP (TO to) (VB substitute)
            (NP (NNS materials))
            (PP (IN in)
              (NP (DT the) (NNP DSTdeal))))))
          (CC or)
          (VP (TO to) (VB modify)
            (NP (NNS materials))
            (PP (IN in)
              (NP (DT the) (NNP DSTdeal)))))))))
    (. .)))

```

```

(ROOT
  (S
    (NP (NNP DSTgdc))
    (VP (VBZ provides)
      (S
        (NP (NNS possibilities))
        (VP
          (VP (TO to) (VB substitute)
            (NP (NNS materials))
            (PP (IN in)
              (NP (DT the) (NNP DSTdeal))))))
          (CC or)
          (VP (TO to) (VB modify)
            (NP (NNS materials))
            (PP (IN in)
              (NP (DT the) (NNP DSTdeal)))))))))
    (. .)))

```

### B.2.16 Remove S#1

Pattern: !ROOT < (S=exc <1 VP !\$- /CC|,/) )

Operation: [excise exc exc]

Example: *Complex VP#1* is already applied

```

(ROOT
  (S
    (NP (NNP DSTGDC))
    (VP (VBZ allows)
      (S
        (VP (TO to) (VB clone)
          (NP (NNP DSTsystems))))))
    (. .)))

(ROOT
  (S
    (NP (NNP DSTGDC))
    (VP (VBZ allows)
      (VP (TO to) (VB clone)
        (NP (NNP DSTsystems))))))
    (. .)))

```

### B.2.17 Remove S#2

Pattern: ROOT < (S < (S=exc <+(S) (SBAR <1 IN)))

Operation: [excise exc exc]

Example: *Complex VP#1* and *Cleanse ADJP* are already applied

```

(ROOT
  (S
    (S
      (SBAR (IN once)
        (S
          (NP (DT the) (NNPS QDstatus))
          (VP (VBZ is) (VBN selected))))
      (, ,)
      (NP (DT the) (NNP DSTscenario))
      (VP (VBZ is) (VBN locked)))
    (CC and)
    (S
      (NP (NNS changes))
      (VP (VBP are) (RB not) (JJ possible)))
    (. .)))

(ROOT
  (S
    (SBAR (IN once)
      (S
        (NP (DT the) (NNPS QDstatus))
        (VP (VBZ is) (VBN selected))))
    (, ,)
    (NP (DT the) (NNP DSTscenario))
    (VP (VBZ is) (VBN locked))
    (CC and)
    (S
      (NP (NNS changes))
      (VP (VBP are) (RB not) (JJ possible)))
    (. .)))

```

### B.2.18 SBAR to VPH

Pattern: VP < (SBAR=rel < (IN < that))

Operation: [relabel rel VPH]

Example:

<pre>(ROOT (S (NP (NNP DSTGDC)) (VP (VBZ assumes) (SBAR (IN that) (S (NP (DT a) (NNP DSTcustomer)) (VP (VBZ is) (VBN assigned) (PP (TO to) (NP (DT an) (NNP opportunity)))) (. .)))</pre>	<pre>(ROOT (S (NP (NNP DSTGDC)) (VP (VBZ assumes) (VPH (IN that) (S (NP (DT a) (NNP DSTcustomer)) (VP (VBZ is) (VBN assigned) (PP (TO to) (NP (DT an) (NNP opportunity)))) (. .)))</pre>
---	--

### B.2.19 SBAR to VPC#1

Pattern: VP <+(SBAR) (SBAR=rel < (IN < /if|whether|after|before/))

Operation: [relabel rel VPC]

Example: *Complex VP#1* is already applied

<pre>(ROOT (S (NP (DT the) (NNP DSTadminDSTtool)) (VP (VBZ allows) (S (VP (TO to) (VB configure) (SBAR (IN whether) (S (NP (NNP DSTcustomers)) (VP (VBP are) (VBN created) (PP (IN by) (NP (NNP DSTgdc)))))))))) (. .)))</pre>	<pre>(ROOT (S (NP (DT the) (NNP DSTadminDSTtool)) (VP (VBZ allows) (S (VP (TO to) (VB configure) (VPC (IN whether) (S (NP (NNP DSTcustomers)) (VP (VBP are) (VBN created) (PP (IN by) (NP (NNP DSTgdc)))))))))) (. .)))</pre>
--	---

### B.2.20 SBAR to VPC#2

Pattern: VP < (SBAR=rel <+(SBAR) (WHADVP < WRB))

Operation: [relabel rel VPC]

Example: *Complex VP#1* is already applied

<pre>(ROOT (S (NP (NNS DSTenvironmentDSTsettings)) (VP (VBP describe) (SBAR (WHADVP (WRB how)) (S (NP (DT the) (NNP DSTcustomer)) (VP (VBZ handles) (NP (DT the) (NN DSTanalyzer)))))) (. .)))</pre>	<pre>(ROOT (S (NP (NNS DSTenvironmentDSTsettings)) (VP (VBP describe) (VPC (WHADVP (WRB how)) (S (NP (DT the) (NNP DSTcustomer)) (VP (VBZ handles) (NP (DT the) (NN DSTanalyzer)))))) (. .)))</pre>
--	---

### B.2.21 SBAR to VPP

Pattern: VP < (SBAR=rel < (IN !< /if|whether|after|before/))

Operation: [relabel rel VPC]

Example:

<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTgrid))     (VP (VBZ offers)       (NP (JJ additional) (NN functionality))       (SBAR (IN as)         (S           (VP (VBD described))))))     (. .)))</pre>	<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTgrid))     (VP (VBZ offers)       (NP (JJ additional) (NN functionality))       (VPP (IN as)         (S           (VP (VBD described))))))     (. .)))</pre>
---	--

### B.2.22 VP to VPV

Pattern: P !<1 /VP.?[VP/ < (VP=rel !< TO !< VP)

Operation: [relabel rel VPV]

Example: *Complex VP#1* is already applied

<pre>(ROOT   (S     (VP (VB click)       (S         (NP (DT the) (NN button))         (VP (TO to)           (VP (VB open)             (NP (DT the) (NNP QDcustQDsearch))             (CC and)             (VP (VB select)               (NP (DT a) ((NN DSTcustomer))))))))))</pre>	<pre>(ROOT   (S     (VP (VB click)       (S         (NP (DT the) (NN button))         (VP (TO to)           (VPV (VB open)             (NP (DT the) (NNP QDcustQDsearch))             (CC and)             (VPV (VB select)               (NP (DT a) ((NN DSTcustomer))))))))))</pre>
---	---

### B.2.23 PP to VPP#1

Pattern: /~VP\$|VPV|NPV/ <+(PP) (PP=rel !< /VB.?!< PP)

Operation: [relabel rel VPP]

Example: *Complex VP#1* is already applied

<pre>(ROOT  (S   (NP    (NP (DT the) (NN number))    (PP (IN of)     (NP (NNP DSTtestDSTreruns))))   (VP (VBZ is) (VBN considered)    (PP (IN during)     (NP (NNP DSTquantification))))   (. .)))</pre>	<pre>(ROOT  (S   (NP    (NP (DT the) (NN number))    (PP (IN of)     (NP (NNP DSTtestDSTreruns))))   (VP (VBZ is) (VBN considered)    (VPP (IN during)     (NP (NNP DSTquantification))))   (. .)))</pre>
--	---

### B.2.24 PP to VPP#2

Pattern: PP=rel \$- /VB.?[MD/

Operation: [relabel rel VPP]

Example: *Complex VP#1* and *Complex VP#3* are already applied

<pre>(ROOT  (S   (NP (DT an) (NNP DSToptimizationDSTmode))   (VP (MD might) (VB be) (JJ available)    (PP (IN for)     (NP (JJ low) (NNP DSTworkload))))   (. .)))</pre>	<pre>(ROOT  (S   (NP (DT an) (NNP DSToptimizationDSTmode))   (VP (MD might) (VB be) (JJ available)    (VPP (IN for)     (NP (JJ low) (NNP DSTworkload))))   (. .)))</pre>
--	---

### B.2.25 VP to VPT#1

Pattern: VP < (VP=rel <+(S) TO)

Operation: [relabel rel VPT]

Example: *Complex VP#1* and *Cleanse S#3* are already applied

<pre>(ROOT  (S   (NP (NNP DSTgdc))   (VP (VBZ allows)    (VP (TO to) (VB configure)     (NP (NNS DSTenvironmentDSTsettings)))   (. .)))</pre>	<pre>(ROOT  (S   (NP (NNP DSTgdc))   (VP (VBZ allows)    (VPT (TO to) (VB configure)     (NP (NNS DSTenvironmentDSTsettings))))   (. .)))</pre>
---	---

### B.2.26 VP to VPT#2

Pattern: /VP\$/=par <+(S) (VP=rel < TO)

Operation: [relabel rel VPT][move rel >1 par]

Example: *Complex VP#1* and *Cleanse S#3* are already applied

<pre>(ROOT   (S     (NP (NNP DSTgdc))     (VP (VBZ allows)       (NP (DT the) (NN user))       (VP (TO to) (VB run)         (NP (DT the) (NN DSTquantificator))))     (. .)))</pre>	<pre>(ROOT   (S     (NP (NNP DSTgdc))     (VP (VBZ allows)       (NP (DT the) (NN user))       (VPT (TO to) (VB run)         (NP (DT the) (NN DSTquantificator))))     (. .)))</pre>
---	--

### B.2.27 VP to VPT#3

Pattern: SBAR <-1 (VP=rel <1 TO)

Operation: [relabel rel VPT]

Example: *Complex VP#1* and *Cleanse S#3* are already applied

<pre>(ROOT   (SBAR (IN in) (NN order)     (VP (TO to) (VB change)       (NP (DT the) (NN DSTprofile))))   (, ,)   (VP (VB select)     (NP (DT an) (NN option)))   (. .))</pre>	<pre>(ROOT   (SBAR (IN in) (NN order)     (VPT (TO to) (VB change)       (NP (DT the) (NN DSTprofile))))   (, ,)   (VP (VB select)     (NP (DT an) (NN option)))   (. .))</pre>
--	---

### B.2.28 VP to VPW

Pattern: /VP\$|VPT|NPV|NPT|PPV|VPV/ < (SBAR=rel < WHNP)

Operation: [relabel rel VPW]

Example: *Complex VP#1* and *Cleanse S#3* are already applied

<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTlist))     (VP (VBZ shows)       (SBAR         (WHNP (WDT which))         (NP (NNP DSTmaterials))         (VP (VBP are) (VBN used)           (PP (IN for)             (NP (NNP DSTquantification))))))     (. .)))</pre>	<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTlist))     (VP (VBZ shows)       (VPW         (WHNP (WDT which))         (NP (NNP DSTmaterials))         (VP (VBP are) (VBN used)           (PP (IN for)             (NP (NNP DSTquantification))))))     (. .)))</pre>
--	---

### B.2.29 PP to NPP

Pattern: /NP\$|PPN|NPP|NPV/ \$+ (PP=rel !< /VB.?/) > /~S\$|^NP\$|PPN/



Operation: [relabel rel NPP]

Example: *Complex VP#1* and *Cleanse S#3* are already applied

<pre>(ROOT   (S     (NP (NNP DSTgdc))     (VP (VBZ performs)       (NP         (NP (CD 3) (NNS steps))         (PP (IN in)           (NP (NNPS DSTquantification))))))     (. .)))</pre>	<pre>(ROOT   (S     (NP (NNP DSTgdc))     (VP (VBZ performs)       (NP         (NP (CD 3) (NNS steps))         (NPP (IN in)           (NP (NNPS DSTquantification))))))     (. .)))</pre>
--	---

### B.2.30 SBAR to NPW

Pattern: NP \$++ (SBAR=rel < WHNP)

Operation: [relabel rel NPW]

Example: *Complex VP#1* is already applied

<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTlist))     (VP (VBZ contains)       (NP         (NP (DT the) (NNPS DSTmaterials))         (SBAR           (WHNP (WDT which))           (S             (VP (VBP are) (VBN used)               (PP (IN for)                 (NP (NNP DSTquantification))))))           (. .)))       (. .)))</pre>	<pre>(ROOT   (S     (NP (DT the) (NNP DSTmaterialDSTlist))     (VP (VBZ contains)       (NP         (NP (DT the) (NNPS DSTmaterials))         (NPW           (WHNP (WDT which))           (S             (VP (VBP are) (VBN used)               (PP (IN for)                 (NP (NNP DSTquantification))))))           (. .)))       (. .)))</pre>
--	---

### B.2.31 VP to NPV

Pattern: (VP=rel [<1 /VB.\*/ | <1 (ADVP \$+ /VB.\*/) | <1 (RB \$+ /VBN|VBD/)] \$- /NP.\*/) > NP

Operation: [relabel rel NPV]

Example:

<pre>(ROOT  (S   (NP (DT the) (NNP DSTquantificator))   (VP (VBZ considers)    (NP     (NP (DT the) (NNS settings))     (VP (VBN made)      (PP (IN on)       (NP (NNP DSTenvt) (NN level))))     (. .)))</pre>	<pre>(ROOT  (S   (NP (DT the) (NNP DSTquantificator))   (VP (VBZ considers)    (NP     (NP (DT the) (NNS settings))     (NPV (VBN made)      (PP (IN on)       (NP (NNP DSTenvt) (NN level))))     (. .)))</pre>
---	--

### B.2.32 NP to PPN

Pattern: /<sup>^</sup>PP\$|VPP|NPP|PPP/ < NP=rel

Operation: [relabel rel PPN]

Example:

<pre>(ROOT  (S   (NP    (NP (DT the) (NN result))    (PP (IN of)     (NP (DT the) (NNP DSTquantificator)))   (VP (VBZ is)    (NP (DT a) (NN DSTmaterialDSTlist)))   (. .)))</pre>	<pre>(ROOT  (S   (NP    (NP (DT the) (NN result))    (PP (IN of)     (PPN (DT the) (NNP DSTquantificator)))   (VP (VBZ is)    (NP (DT a) (NN DSTmaterialDSTlist)))   (. .)))</pre>
---	--

### B.2.33 PP to PPV

Pattern: /<sup>^</sup>PP\$|VPP|NPP|PPP/ < VP=rel

Operation: [relabel rel PPV]

Example: *Cleanse S#3* is already applied

<pre>(ROOT  (S   (VP (VB exclude)    (NP (NNS materials))    (PP (IN before)     (VP (VBG running)      (NP (DT the) (NN DSTquantificator))))</pre>	<pre>(ROOT  (S   (VP (VB exclude)    (NP (NNS materials))    (PP (IN before)     (PPV (VBG running)      (NP (DT the) (NN DSTquantificator))))</pre>
---	--

### B.2.34 PP to PPW#1

Pattern: /<sup>^</sup>PP\$|VPP|NPP|PPP/ < (IN \$+ (SBAR=rel < (WHNP < WDT)))

Operation: [relabel rel PPW]

Example: *Cleanse S#3* is already applied

```
(ROOT
(S
(NP (DT the) (NNP QDorigin) (NN column))
(VP (VBZ shows)
(NP (DT the) (NN source))
(PP (IN from)
(SBAR
(WHNP (WDT which))
(NP (DT the) (NN material))
(VP (VBD was) (VBN added)
(PP (TO to)
(NP (DT the) (NNP DSTscenario))))
(. .)))
```

```
(ROOT
(S
(NP (DT the) (NNP QDorigin) (NN column))
(VP (VBZ shows)
(NP (DT the) (NN source))
(PP (IN from)
(PPW
(WHNP (WDT which))
(NP (DT the) (NN material))
(VP (VBD was) (VBN added)
(PP (TO to)
(NP (DT the) (NNP DSTscenario))))
(. .)))
```

### B.2.35 PP to PPW#2

Pattern: SBAR=rel1 < (WHPP=rel2 < IN=mov1 < WHNP=par \$+ S=mov2)

Operation: [move mov1 >1 rel1][move mov2 par][relabel rel1 NPP][relabel rel2 PPW]

Example: *Complex VP#1* is already applied

<pre>(ROOT (S (NP (NP (DT the) (NNPS DSTsources)) (SBAR (WHPP (IN from) (WHNP (WDT which))) (S (NP (DT the) (NNPS DSTsettings)) (VP (VBD were) (VBN loaded)))))) (VP (VBP are) (VBN displayed)) (. .)))</pre>	<pre>(ROOT (S (NP (NP (DT the) (NNPS DSTsources)) (NPP (PPW (IN from) (WHNP (WDT which))) (S (NP (DT the) (NNPS DSTsettings)) (VP (VBD were) (VBN loaded)))))) (VP (VBP are) (VBN displayed)) (. .)))</pre>
---	---

### B.2.36 Surround NP

Pattern: /<sup>^</sup>NP\$|PPN\$/=sis \$+ /<sup>^</sup>PP\$|NPP\$/ \$++ \_\_\_\_\_=bro !> /<sup>^</sup>NP\$|PPN\$/ > \_\_\_\_\_=par

Operation: [move mov1 >1 rel1][move mov2 par][relabel rel1 NPP][relabel rel2 PPW]

Example: *Complex VP#1* is already applied

<pre>(ROOT   (S     (NP       (NP (DT the) (NN DSTlog))       (PP (IN for)         (NP (NMP DSTtest))))     (VP (VBZ comprises)       (NP (JJ detailed) (NN information))       (PP (IN about))))))</pre>	<pre>(ROOT   (S     (NP       (NP (DT the) (NN DSTlog))       (PP (IN for)         (NP (NMP DSTtest))))     (VP (VBZ comprises)       (NP         (NP (JJ detailed) (NN information))         (PP (IN about))))))</pre>
---	---

# Publications

Parts of the literature review, formal concepts and evaluation results of this thesis are already published as scientific publications. In the following, an overview of the relevant publications in chronological order is provided:

1. T. Quirchmayr, B. Paech, H. Karey, R. Kohl: **Semi-automatic Rule-based Domain Terminology and Software Feature-relevant Information Extraction from Natural Language User Manuals**. Empirical Software Engineering x(y), 2018.
2. T. Quirchmayr, B. Paech, H. Karey, R. Kohl: **Semi-automatic Software Feature-Relevant Information Extraction from Natural Language User Manuals** In: Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17), Essen, March 2017.

# Bibliography

- Aggarwal, C. C. (2015). *Data Mining: The Textbook*. Springer Publishing Company.
- Aggarwal, C. C., and Zhai, C. (2012a). *Mining Text Data*. Springer-Verlag New York.
- Aggarwal, C. C., and Zhai, C. (2012b). A survey of text classification algorithms. In *Mining text data*, (pp. 163–222). Springer.
- Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., and Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*.
- Apel, S., and Kästner, C. (2009). An overview of feature-oriented software development. *The Journal of Object Technology*, 8(5), 49–84.
- Bakar, N. H., Kasirun, Z. M., and Salleh, N. (2015). Terms extractions: An approach for requirements reuse. In *2nd International Conference on Information Science and Security (ICISS '15)*, (pp. 1–4).
- Bakar, N. H., Kasirun, Z. M., Salleh, N., and Jalab, H. A. (2016). Extracting features from online software reviews to aid requirements reuse. *Applied Software Computing*, 49, 1297–1315.
- Balachandran, K., and Ranathunga, S. (2016). Domain-specific term extraction for concept identification in ontology construction. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, (pp. 34–41).
- Bender, E. M. (2013). *Linguistic Fundamentals for Natural Language Processing: 100 Essentials from Morphology and Syntax*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- Berry, D., Gacitua, R., Sawyer, P., and Tjong, S. F. (2012). *The Case for Dumb Requirements Engineering Tools*, (pp. 211–217). Springer.

- Bezdek, J. C., Ehrlich, R., and Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3), 191–203.
- Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.
- Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(1), 993–1022.
- Boonthum-Denecke, C. (2011). *Cross-Disciplinary Advances in Applied Natural Language Processing: Issues and Approaches*. IGI Global.
- Bosch, J. (2000). *Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach*. ACM Press/Addison-Wesley Publishing Co.
- Bourque, P., and Fairley, R. E. (2014). *Guide to the software engineering body of knowledge (SWEBOOK (R)): Version 3.0*. IEEE Computer Society Press.
- Boutkova, E., and Houdek, F. (2011). Semi-automatic identification of features in requirement specifications. In *19th IEEE International Requirements Engineering Conference (RE '11)*, (pp. 313–318).
- Brinton, L. J., and Brinton, D. (2010). *The linguistic structure of modern English*. John Benjamins Publishing.
- Castañeda, V., Ballejos, L., Caliusco, M. L., and Galli, M. R. (2010). The use of ontologies in requirements engineering. *Global Journal of Research In Engineering*, 10(6).
- Celeux, G., and Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern recognition*, 28(5), 781–793.
- Chandrasekar, R., Doran, C., and Srinivas, B. (1996). Motivations and methods for text simplification. In *Proceedings of the 16th conference on Computational linguistics*, (pp. 1041–1044).
- Chen, J., Chau, R., and Yeh, C.-H. (2004). Discovering parallel text from the world wide web. In *Proceedings of the Second Workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation (ACSW Frontiers '04)*, (pp. 157–161).
- Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1), 51–89.
- Classen, A., Heymans, P., and Schobbens, P.-Y. (2008). What’s in a feature: A requirements engineering perspective. In *Proceedings of the 11th International Conference on*

- Fundamental Approaches to Software Engineering, Theory and Practice of Software*, (pp. 16–30).
- Colas, F., and Brazdil, P. (2006). Comparison of SVM and some older classification algorithms in text classification tasks. In *Proceedings of the International Conference on Artificial Intelligence in Theory and Practice (IFIP'06)*, (pp. 169–178).
- Corbett, G. (2006). Linguistic features. *African Affairs*, 87, 25–54.
- Dale, R., Moisl, H., and Somers, H. (2000). *Handbook of natural language processing*. CRC Press.
- Dong, Y.-S., and Han, K.-S. (2004). A comparison of several ensemble methods for text categorization. In *Proceedings of the IEEE International Conference on Services Computing (SCC'04)*, (pp. 419–422).
- Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., and Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. In *33rd International Conference on Software Engineering (ICSE'11)*, (pp. 181–190).
- Earls, A., Embury, S., and Turner, N. (2002). A method for the manual extraction of business rules from legacy source code. *BT Technology*, 20(4), 127–145.
- Eisenbarth, T., Koschke, R., and Simon, D. (2003). Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3), 210–224.
- Feldman, R., and Dagan, I. (1995). Knowledge discovery in textual databases (KDT). In *KDD*, vol. 95, (pp. 112–117).
- Feldman, S. (1999). NLP meets the jabberwocky: Natural language processing in information retrieval. *Online-Weston*, 23, 62–73.
- Feldt, R., and Magazinius, A. (2010). Validity threats in empirical software engineering research—an initial survey. In *Proceedings of the Software Engineering and Knowledge Engineering Conference (SEKE'10)*, (pp. 374–379).
- Forward, A., and Lethbridge, T. C. (2002). The relevance of software documentation, tools and technologies: a survey. *Proceedings of the ACM symposium on Document engineering*, (pp. 26–33).
- Frakes, W. B., and Baeza-Yates, R. (1992). *Information retrieval: Data structures & algorithms*. Prentice Hall Englewood Cliffs.



- Fuller, R. B. (1957). A comprehensive anticipatory design science. *Journal of Royal Architectural Institute of Canada*, (pp. 84–117).
- Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.
- Gantz, J., and Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future, 2007*(2012), 1–16.
- Ghosh, J., and Strehl, A. (2006). Similarity-based text clustering: A comparative study. In J. Kogan, C. Nicholas, and M. Teboulle (Eds.) *Grouping Multidimensional Data: Recent Advances in Clustering*, (pp. 73–97). Springer Berlin Heidelberg.
- Ghosh, S., Elenius, D., Li, W., Lincoln, P., Shankar, N., and Steiner, W. (2014). ARSENAL: Automatic requirements specification extraction from natural language. *arXiv preprint arXiv:1403.3142*.
- Godfrey, M. W., and German, D. M. (2008). The past, present, and future of software evolution. In *Frontiers of Software Maintenance (FoSM'08)*, (pp. 129–138).
- Grefenstette, G., and Tapanainen, P. (1994). What is a word, what is a sentence?: problems of tokenisation.
- Guzman, E., and Maalej, W. (2014). How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the 22nd International Requirements Engineering Conference (RE'14)*, (pp. 153–162).
- Hancke, J., Vajjala, S., and Meurers, D. (2012). Readability classification for german using lexical, syntactic, and morphological features. *Proceedings of COLING 2012*, (pp. 1063–1080).
- Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., and Mobasher, B. (2013). Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 39(12), 1736–1752.
- Hassan, S., Rafi, M., and Shaikh, M. S. (2011). Comparing svm and naive bayes classifiers for text categorization with wikitology as knowledge enrichment. In *IEEE 14th International Multitopic Conference (INMIC)*, (pp. 31–34).
- Hotho, A., a. Nürnberger, and Paaß, G. (2005). A brief survey of text mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, 20(1), 19–62.

- Hsieh, S.-H., Lin, H.-T., Chi, N.-W., Chou, K.-W., and Lin, K.-Y. (2011). Enabling the development of base domain ontology through extraction of knowledge from engineering domain handbooks. *Advanced Engineering Informatics*, 25(2), 288–296.
- IEEE (1990). IEEE standard glossary of software engineering terminology. *Office*, 121990(1), 1.
- Johann, T., Stanik, C., Alizadeh, A. M., and Maalej, W. (2017). SAFE: A simple approach for feature extraction from app descriptions and app reviews. In *IEEE 25th International Requirements Engineering Conference (RE'17)*, (pp. 21–30).
- John, I. (2010). Using documentation for product line scoping. *IEEE Software*, 27(3), 42–47.
- Jonnalagadda, S., Tari, L., Hakenberg, J., Baral, C., and Gonzalez, G. (2009). Towards effective sentence simplification for automatic processing of biomedical text. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, (pp. 177–180). Association for Computational Linguistics.
- Jorgensen, M., and Shepperd, M. (2007). A systematic review of software development cost estimation studies. *IEEE Transactions of Software Engineering*, 33(1), 33–53.
- Jurafsky, D., and Martin, J. H. (2014). *Speech and language processing*. Pearson.
- Khan, K., Baharudin, B., and Khan, A. (2014). Identifying product features from customer reviews using hybrid patterns. *International Arab Journal of Information Technology*, 11(3), 281–286.
- Kim, S. N., Baldwin, T., and Kan, M.-Y. (2009). An unsupervised approach to domain-specific term extraction. In *Australasian Language Technology Association Workshop 2009*, (pp. 94–98).
- Kim, S. N., and Cavedon, L. (2011). Classifying domain-specific terms using a dictionary. In *Australasian Language Technology Association Workshop 2011*, (pp. 57–65).
- Kitchenham, B., and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Keele University and Durham University Joint Report.
- Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5), 604–632.

- Langley, P., Iba, W., and Thompson, K. (1992). An analysis of Bayesian classifiers. *Artificial Intelligence*, (pp. 223–228).
- Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, vol. 10, (pp. 707–710).
- Levy, R., and Andrew, G. (2006). Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC'06)*, (pp. 2231–2234).
- Li, Y., Guzman, E., Tsiamoura, K., Schneider, F., and Bruegge, B. (2015). Automated requirements extraction for scientific software. *Procedia Computer Science*, 51, 582–591.
- Liddy, E. D. (1998). Enhanced text retrieval using natural language processing. *Bulletin of the American Society for Information Science and Technology*, 24(4), 14–16.
- Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge university press.
- Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and 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).
- Marcuska, S., Gencel, C., and Abrahamsson, P. (2014). Automated feature identification in web applications. In *Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering*, (pp. 100–114). Springer.
- Mitkov, R. (2014). *Anaphora resolution*. Routledge.
- Nixon, M. (2008). *Feature extraction & image processing*. Academic Press.
- Noy, N. F., and McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology.
- Paech, B., Hübner, P., and Merten, T. (2014). What Are the Features of This Software ? An Exploratory Study. In *Proceedings of the 9th International Conference on Software Engineering Advances*.
- Petticrew, M., and Roberts, H. (2006). *Systematic Reviews in the Social Sciences: A Practical Guide*. Blackwell Pub.
- Phan, X.-H., Nguyen, L.-M., and Horiguchi, S. (2008). Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, (pp. 91–100).

- Piatetski, G., and Frawley, W. (1991). *Knowledge discovery in databases*. MIT press.
- Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering Journal*, 13(3), 303–337.
- Rajman, M., and Besançon, R. (1998). Text mining: natural language techniques and text mining applications. In *Data mining and reverse engineering*, (pp. 50–64).
- Rangrej, A., Kulkarni, S., and Tendulkar, A. V. (2011). Comparative study of clustering techniques for short text documents. In *Proceedings of the 20th international conference companion on World wide web*, (pp. 111–112).
- Shaker, P., Atlee, J. M., and Wang, S. (2012). A feature-oriented requirements modelling language. In *Proceedings of 20th International Requirements Engineering Conference (RE'12)*, (pp. 151–160).
- Slankas, J., and Williams, L. (2013). Automated extraction of non-functional requirements in available documentation. In *1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE'13)*, (pp. 9–16).
- Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, (pp. 173–180). Association for Computational Linguistics.
- Toutanova, K., and Manning, C. D. (2000). Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, (pp. 63–70). Association for Computational Linguistics.
- Uysal, A. K., and Gunal, S. (2014). The impact of preprocessing on text classification. *Information Processing & Management*, 50(1), 104–112.
- Venu, S. H., Mohan, V., Urkalan, K., and Geetha, T. V. (2016). Unsupervised domain ontology learning from text. In *Proceedings of the International Conference on Mining Intelligence and Knowledge Exploration*, (pp. 132–143).
- Vijayarani, S., Ilamathi, J., and Nithya, M. (2015). Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1), 7–16.

- Ward, L. J., and Woods, G. (2013). *English grammar for dummies*. John Wiley & Sons.
- Weston, N., Chitchyan, R., and Rashid, A. (2009). A framework for constructing semantically composable feature models from natural language requirements. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*, (pp. 211–220).
- Wieringa, R., and Morali, A. (2012). Technical action research as a validation method in information systems design science. In *Proceedings of the 7th International Conference on Design Science Research in Information Systems: Advances in Theory and Practice (DESRIST'12)*.
- Wimalasuriya, D. C., and Dou, D. (2010). Ontology-based information extraction: An introduction and a survey of current approaches. *Journal of Information Science*, (pp. 306–323).
- Wohlin, C., M., P. R., Höst, Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.
- Yu, Y., Wang, H., Yin, G., and Liu, B. (2013). Mining and recommending software features across multiple web repositories. In *Proceedings of the 5th Asia-Pacific Symposium on Internetware*, (pp. 9–15).
- Zhan, T.-J., and Li, C.-H. (2010). Product feature mining with nominal semantic structure. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'10)*, (pp. 464–467).
- Zorn-Pauli, G., Paech, B., and Wittkopf, J. (2012). Strategic release planning challenges for global information systems - a position paper. In *Proceedings of the 6th International Workshop on Software Product Management (IWSPM'12)*, (pp. 186–191).