



UNIVERSITÉ DU  
LUXEMBOURG

PhD-FSTC-2016-44  
The Faculty of Sciences, Technology and Communication

## DISSERTATION

Presented on 14/10/2016 in Luxembourg  
to obtain the degree of

### DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE

by  
**CHETAN ARORA**

Born on 27th October 1985 in Ludhiana (Punjab, India)

### AUTOMATED ANALYSIS OF NATURAL-LANGUAGE REQUIREMENTS USING NATURAL LANGUAGE PROCESSING

#### DISSERTATION DEFENSE COMMITTEE

DR.-ING. LIONEL BRIAND, Dissertation Supervisor  
*University of Luxembourg (Luxembourg)*

DR.-ING. YVES LE TRAON, Chairman  
*University of Luxembourg (Luxembourg)*

DR. SHIVA NEJATI, Vice-Chairman  
*University of Luxembourg (Luxembourg)*

PROF. DR. TONY GORSCHEK, Member  
*Blekinge Institute of Technology (Sweden)*

PROF. DR. PETE SAWYER, Member  
*Lancaster University (UK)*

DR. MEHRDAD SABETZADEH, Expert (Dissertation Co-supervisor)  
*University of Luxembourg (Luxembourg)*



## Abstract

Natural Language (NL) is arguably the most common vehicle for specifying requirements. This dissertation devises automated assistance for some important tasks that requirements engineers need to perform in order to structure, manage, and elaborate NL requirements in a sound and effective manner. The key enabling technology underlying the work in this dissertation is Natural Language Processing (NLP). All the solutions presented herein have been developed and empirically evaluated in close collaboration with industrial partners.

The dissertation addresses four different facets of requirements analysis:

- **Checking conformance to templates.** Requirements templates are an effective tool for improving the structure and quality of NL requirements statements. When templates are used for specifying the requirements, an important quality assurance task is to ensure that the requirements conform to the intended templates. We develop an *automated solution for checking the conformance of requirements to templates*.
- **Extraction of glossary terms.** Requirements glossaries (dictionaries) improve the understandability of requirements, and mitigate vagueness and ambiguity. We develop an *automated solution for supporting requirements analysts in the selection of glossary terms and their related terms*.
- **Extraction of domain models.** By providing a precise representation of the main concepts in a software project and the relationships between these concepts, a domain model serves as an important artifact for systematic requirements elaboration. We propose an *automated approach for domain model extraction from requirements*. The extraction rules in our approach encompass both the rules already described in the literature as well as a number of important extensions developed in this dissertation.
- **Identifying the impact of requirements changes.** Uncontrolled change in requirements presents a major risk to the success of software projects. We address two different dimensions of requirements change analysis in this dissertation: First, we develop an *automated approach for predicting how a change to one requirement impacts other requirements*. Next, we consider the propagation of change from requirements to design. To this end, we develop an *automated approach for predicting how the design of a system is impacted by changes made to the requirements*.



## Acknowledgments

This dissertation is the outcome of a journey that started in 2012. Reminiscing the past four years of this incredible journey, a number of people come to my mind that have supported and encouraged me to reach this milestone.

First of all, I would like to thank my supervisor, Lionel Briand, for giving me the opportunity to work with him, and for being a constant source of inspiration in each step of the way. This journey would not have been possible without his support and guidance.

I would like to thank my co-supervisor, Mehrdad Sabetzadeh, who is an epitome of a great mentor and researcher. I have learned a great deal of things both professionally and personally under his supervision.

I am grateful to the members of my defense committee: Yves Le Traon, Shiva Nejati, Tony Gorschek, and Pete Sawyer for their insightful suggestions and remarks. I would like to additionally thank Tony Gorschek and Pete Sawyer for taking the time to travel to Luxembourg to attend my defense, despite their busy schedules.

I would like to express my gratitude to SES Techcom for allowing me to gain first-hand insights into real-world challenges in requirements analysis, and for providing case study material. In particular, I would like to thank Frank Zimmer from SES for always being ready to help and for being such a great mentor.

I also appreciate all the helpful advice and feedback I received from my colleagues in the Software Verification and Validation Lab. I thank them all for offering a warm and welcoming working environment.

Finally, I would like to thank my entire family in India. My wife encouraged and motivated me to undertake this journey and was extremely patient and supportive throughout despite her own Ph.D. journey in parallel. I would also like to thank my parents for always believing in me and taking pride in everything I do.



# Contents

<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xiii</b>
<b>Acronyms</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context . . . . .	1
1.2 Contributions and Organization . . . . .	2
<b>2 Background</b>	<b>5</b>
2.1 Natural Language Requirements . . . . .	5
2.2 Natural Language Processing . . . . .	6
2.3 UML and SysML . . . . .	8
2.4 Quality Attributes . . . . .	10
<b>3 Requirements Template Conformance Checking</b>	<b>11</b>
3.1 Motivation and Contributions . . . . .	12
3.2 Background . . . . .	14
3.2.1 Requirements Templates . . . . .	14
3.2.2 Text Chunking . . . . .	15
3.2.3 Pattern Matching in NLP . . . . .	17
3.3 Approach . . . . .	18
3.3.1 Expressing Templates as Grammars . . . . .	18
3.3.2 Conformance Checking via Pattern Matching . . . . .	19
3.3.3 Handling Complex Phrases . . . . .	23
3.3.4 Checking NL Best Practices . . . . .	24
3.4 Tool Support . . . . .	24
3.5 Evaluation . . . . .	27
3.5.1 Research Questions (RQs) . . . . .	27
3.5.2 Description of Case Studies . . . . .	28
3.5.3 Data Collection Procedure . . . . .	29
3.5.4 Analysis Procedure . . . . .	30
3.5.4.1 NLP Pipeline Configuration . . . . .	30
3.5.4.2 Metrics for Measuring Accuracy . . . . .	32

3.5.5	Results . . . . .	33
3.5.5.1	Requirements Inspection and Glossary Elicitation . . . . .	33
3.5.5.2	Accuracy and Execution Time . . . . .	35
3.5.6	Discussion . . . . .	36
3.6	Limitations and Threats to Validity . . . . .	43
3.7	Related Work . . . . .	44
3.7.1	Requirements Templates . . . . .	44
3.7.2	NLP in Requirements Engineering . . . . .	45
3.8	Conclusion . . . . .	46
<b>4</b>	<b>Glossary Terms Extraction and Clustering</b>	<b>47</b>
4.1	Motivation and Contributions . . . . .	48
4.2	Background . . . . .	50
4.2.1	Text Chunking . . . . .	50
4.2.2	Computing Similarities between Terms . . . . .	51
4.2.2.1	Syntactic Similarity Measures . . . . .	51
4.2.2.2	Semantic Similarity Measures . . . . .	51
4.2.3	Clustering . . . . .	53
4.2.3.1	Clustering Algorithms . . . . .	54
4.2.3.2	Choosing the Number of Clusters . . . . .	55
4.2.4	Related Work . . . . .	55
4.2.4.1	Term Extraction . . . . .	56
4.2.4.2	Clustering . . . . .	57
4.2.4.3	Natural Language Processing . . . . .	58
4.3	Approach . . . . .	59
4.3.1	Extracting Candidate Glossary Terms . . . . .	59
4.3.2	Computing Similarities between Terms . . . . .	60
4.3.3	Clustering Terms . . . . .	60
4.4	Tool Support . . . . .	61
4.5	Evaluation . . . . .	63
4.5.1	Research Questions . . . . .	63
4.5.2	Description of Case Studies . . . . .	64
4.5.3	Case Selection Criteria . . . . .	64
4.5.4	Data Collection Procedure . . . . .	65
4.5.4.1	Glossary Term Elicitation . . . . .	65
4.5.4.2	Domain Model Construction . . . . .	66
4.5.4.3	Expert Interview Survey . . . . .	68
4.5.5	Analysis Procedure . . . . .	69
4.5.5.1	Inferring Ideal Clusters . . . . .	69
4.5.5.2	Evaluation Procedure . . . . .	70
4.5.6	Results and Discussion . . . . .	72
4.5.6.1	RQ1. How accurate is our approach at extracting glossary terms? . . . . .	72
4.5.6.2	RQ2. Which similarity measure(s) and clustering algorithms(s) yield the most accurate clusters? . . . . .	74
4.5.6.3	RQ3. How can one specify the number of clusters? . . . . .	79

4.5.6.4	RQ4. Which of the alternatives identified in RQ2 are the most accurate when used with the guidelines from RQ3? . . . . .	81
4.5.6.5	RQ5. Does our approach run in practical time? . . . . .	82
4.5.6.6	RQ6. How effective is our clustering technique at grouping related terms? . . . . .	84
4.5.6.7	RQ7. Do practitioners find the clusters generated by our approach useful? . . . . .	86
4.6	Threats to Validity . . . . .	87
4.7	Conclusion . . . . .	88
<b>5</b>	<b>Extracting Domain Models from Natural-Language Requirements</b>	<b>89</b>
5.1	Motivation and Contributions . . . . .	90
5.2	State of the Art . . . . .	93
5.3	Syntactic Parsing . . . . .	95
5.4	Approach . . . . .	97
5.4.1	Processing the Requirements Statements . . . . .	97
5.4.2	Deriving Dependencies at a Semantic Level . . . . .	98
5.4.3	Domain Model Construction . . . . .	98
5.5	Empirical Evaluation . . . . .	102
5.5.1	Implementation . . . . .	103
5.5.2	Results and Discussion . . . . .	103
5.5.3	Limitations and Validity Considerations . . . . .	109
5.6	Conclusion . . . . .	110
<b>6</b>	<b>Inter-Requirements Change Impact Analysis</b>	<b>111</b>
6.1	Motivation and Contributions . . . . .	112
6.2	Overview of the Approach . . . . .	114
6.3	Background . . . . .	115
6.4	Processing of Requirements . . . . .	116
6.5	Change Application and Differencing . . . . .	117
6.6	Specification of Propagation Condition . . . . .	118
6.7	Calculation of Impact Likelihoods . . . . .	120
6.8	Tool Support . . . . .	122
6.9	Evaluation . . . . .	123
6.10	Related Work . . . . .	128
6.11	Conclusion . . . . .	129
<b>7</b>	<b>Change Impact Analysis between SysML Models of Requirements and Design</b>	<b>131</b>
7.1	Motivation and Contributions . . . . .	132
7.2	Approach . . . . .	135
7.2.1	Building SysML Models . . . . .	136
7.2.2	Computing Potentially Impacted Elements . . . . .	138
7.2.3	Ranking Potentially Impacted Elements . . . . .	142
7.3	Empirical Evaluation . . . . .	143
7.4	Related Work . . . . .	150
7.5	Conclusion . . . . .	152

*Contents*

---

<b>8 Conclusion</b>	<b>153</b>
8.1 Summary . . . . .	153
8.2 Future Work . . . . .	154
<b>List of Papers</b>	<b>157</b>
<b>Bibliography</b>	<b>159</b>

# List of Figures

1.1	Dissertation overview and organization. . . . .	2
2.1	Relation between SysML and UML [OMG, 2016]. . . . .	8
3.1	Rupp’s template [Pohl and Rupp, 2011]. . . . .	12
3.2	Example requirements: $R_1$ and $R_2$ conform to Rupp’s template but $R_3$ does not. The fixed elements of the template are written in capital letters. . . . .	12
3.3	The EARS template [Mavin et al., 2009]. . . . .	14
3.4	(a) An example requirements statement, (b) its sentence chunks, and (c) its full parse tree. . . . .	16
3.5	NLP pipeline for text chunking. . . . .	16
3.6	JAPE script for Rupp’s Autonomous type. . . . .	17
3.7	BNF grammars for (a) Rupp’s template and (b) the EARS template. . . . .	19
3.8	Pipeline for Template Conformance Checking. . . . .	20
3.9	Annotations generated by the pipeline of Figure 3.8 over the example requirements of Figure 3.2. . . . .	21
3.10	JAPE script for marking details. . . . .	22
3.11	JAPE script for marking conditional details. . . . .	22
3.12	JAPE script for marking the event-driven conditional type in EARS. . . . .	22
3.13	Enhancing TCC with parsing. . . . .	23
3.14	JAPE script for marking complex NPs using a parse tree. . . . .	24
3.15	RETA tool architecture. . . . .	26
3.16	Snapshot of the annotations generated by RETA. . . . .	26
3.17	Confusion matrix for measuring accuracy. . . . .	32
3.18	Box plots for classification accuracy metrics and execution times. . . . .	36
3.19	Accuracy results for Case-D with and without considering complex noun phrases. . . . .	37
3.20	Regression trees for $F_2$ -measure. . . . .	38
3.21	Execution time growth for checking conformance (a) to Rupp’s template and (b) to the EARS template. . . . .	39
3.22	Execution time growth for the Parser step envisaged by the process of Figure 3.13. . . . .	42
4.1	(a) Example requirements from a satellite component, (b) candidate glossary terms extracted from the requirements, (c) candidate terms organized into clusters. . . . .	49
4.2	Example of semantic similarity calculation for multi-word terms. . . . .	53
4.3	Approach overview. . . . .	58
4.4	Tool Overview. . . . .	61
4.5	Screenshot of REGICE (implemented in GATE [GATE, 2016]) with two computed clusters highlighted. . . . .	62

4.6	Description of case studies. . . . .	63
4.7	A fragment of the domain model for Case-A. . . . .	67
4.8	Domain model versus glossary: grayed-out model elements have no corresponding glossary term. . . . .	67
4.9	Statements for assessing the usefulness of a cluster. . . . .	68
4.10	Accuracy of terms extraction. . . . .	73
4.11	<i>F</i> -measure curves for three different cluster computation alternatives. . . . .	75
4.12	Regression trees for normalized AUCs. . . . .	77
4.13	Comparison between the K-means and EM clustering algorithms; both curves are for Case-C and computed using the combination of Levenstein and PATH. . . . .	78
4.14	Selecting the number of clusters using BIC. . . . .	80
4.15	Accuracy of the alternatives identified in RQ2 when the number of clusters is set using the guidelines of RQ3. . . . .	81
4.16	Clustering example: ovals represent ideal clusters and background colors represent generated clusters. . . . .	83
4.17	Comparison with random partitioning; the optimal number of clusters for each case study is further shown. . . . .	83
4.18	Cluster size distributions. . . . .	85
4.19	Expert survey interview results. . . . .	86
5.1	(a) Example requirements statement and (b) corresponding domain model fragment. . . . .	90
5.2	(a) Example relative clause modifier (rcmod) dependency and (b) corresponding relation. . . . .	91
5.3	(a) Example requirements statement, (b) direct relation, (c-d) link path (indirect) relations. . . . .	92
5.4	(a) A requirement and (b) its parse tree. . . . .	95
5.5	Results of dependency parsing for requirement R4 of Figure 5.4(a). . . . .	95
5.6	Parsing pipeline. . . . .	96
5.7	Approach Overview. . . . .	97
5.8	Algorithm for lifting word dependencies to semantic-unit dependencies. . . . .	99
5.9	Interview survey questionnaire. . . . .	105
5.10	(a) Raw and (b) bootstrapping results for Q1 and Q2. . . . .	106
5.11	% of relevant relations retrieved. . . . .	109
6.1	Example requirements from a satellite control system (with changes). Added text is green and underlined. Removed text is red and struck through. . . . .	112
6.2	Approach overview. . . . .	114
6.3	Details of the requirements processing step. . . . .	116
6.4	Differencing example. . . . .	118
6.5	Algorithm for detecting added & deleted phrases (phrasal differencing). . . . .	118
6.6	Grammar for propagation conditions. . . . .	119
6.7	Matching a propagation condition phrase against a requirement. . . . .	120
6.8	NARCIA's user interface for (a) specifying propagation conditions, and (b) reviewing change impact analysis results (tool's output has been truncated). . . . .	122
6.9	Accuracy of lists produced by two combinations of similarity measures. . . . .	125
6.10	Delta chart for identifying the cutoff ( <i>r</i> ). . . . .	126
6.11	FP rates. . . . .	127
7.1	Requirements diagram fragment for CP. . . . .	132

---

7.2	Fragment of CP's block diagram. . . . .	133
7.3	(Simplified) activity diagram for the Diagnostics Manager block ( $B_3$ ) of Figure 7.2. . . . .	133
7.4	Approach overview. . . . .	136
7.5	The traceability information required by our change impact analysis approach. . . . .	137
7.6	Example activity diagram with control input/output and a call action node. . . . .	138
7.7	Algorithm for computing an estimated impact set ( <i>EIS</i> ) using inter-block structural relations.	139
7.8	Algorithm for computing <i>EIS</i> using both structural and behavioral relations. . . . .	140
7.9	Algorithm for activity diagram slicing (used by the algorithm of Figure 7.8). . . . .	141
7.10	Impact of behavioral analysis: Comparing (a) the size of <i>EISs</i> and (b) the precision of <i>EISs</i> obtained by structural analysis alone and by combined behavioral and structural analysis. . . . .	146
7.11	Two alternative applications of similarity measures for ranking the same <i>EIS</i> . . . . .	146
7.12	Regression tree for identifying the best similarity measure alternatives. . . . .	148
7.13	Ranked similarity scores and delta chart for an example change scenario from CP. The delta chart is used for computing the cutoff ( $r$ ). . . . .	148
7.14	Size and precision of <i>EISs</i> that result from the application of the guidelines of RQ3 to the <i>EISs</i> computed by the algorithm of Figure 7.8. . . . .	149



# List of Tables

2.1	Feature list example. . . . .	6
2.2	NLP techniques applied for different approaches in this dissertation. . . . .	7
2.3	SysML Requirement Diagram relationship types. . . . .	9
3.1	Potentially problematic constructs (from Berry et al [Berry et al., 2003]) detected through NLP. . . . .	25
3.2	The case studies used for evaluation. . . . .	28
3.3	General statistics for the case studies. . . . .	33
3.4	Results from the analysis of non-conformant requirements. . . . .	34
3.5	Best pipelines in terms of $F_2$ -measure. . . . .	35
3.6	Average accuracy for recommended pipeline configurations. . . . .	40
3.7	Expected number of false positives and false negatives based on average accuracy levels (Table 3.6). . . . .	41
4.1	Description of the syntactic similarity measures considered in our empirical evaluation. .	52
4.2	Description of the semantic similarity measures considered in our empirical evaluation. .	52
4.3	Description of the alternative criteria considered in our empirical evaluation for computing cluster distances when hierarchical clustering is applied. . . . .	54
4.4	Heuristics applied to the results of text chunking. . . . .	60
4.5	Information about the case studies. . . . .	72
4.6	Top-5 cluster computation alternatives. . . . .	76
4.7	Execution times. . . . .	82
4.8	(a) Upper bounds for the accuracy of partitioning clustering, (b) the actual accuracy of our approach. . . . .	83
5.1	Existing domain model extraction rules. . . . .	94
5.2	New extraction rules in our approach. . . . .	99
5.3	Extraction results for R4 of Figure 5.4(a). . . . .	100
5.4	Different subject types. . . . .	100
5.5	Description of case study documents. . . . .	103
5.6	Number of times extraction rules were triggered and number of extracted elements (per document). . . . .	103
5.7	Correctness and relevance results obtained from our expert interview, organized by extraction rules. . . . .	104
5.8	Reasons for inaccuracies and non-relevance. . . . .	108
6.1	Example propagation conditions and their explanation. . . . .	119

---

*List of Tables*

---

6.2	Case studies used in the evaluation. . . . .	124
6.3	Shapes of propagation conditions and sizes of impact sets. . . . .	124
6.4	Execution Times. . . . .	128
7.1	Additional examples of change statements . . . . .	145

# Acronyms

**AD** Activity Diagram.

**ANNIE** A Nearly-New Information Extraction system.

**API** Application Programming Interface.

**AUC** Area Under the Curve.

**BDD** Block Definition Diagram.

**BIC** Bayesian Information Criterion.

**BNF** Backus-Naur Form.

**CD** Class Diagram.

**CFG** Context-Free Grammar.

**CIA** Change Impact Analysis.

**CPU** Central Processing Unit.

**EA** Enterprise Architect.

**EARS** Easy Approach to Requirements Syntax.

**EIS** Estimated Impact Set.

**ESA** European Space Agency.

**GATE** General Architecture for Text Engineering.

**GUI** Graphical User Interface.

**HTML** HyperText Markup Language.

**IBD** Internal Block Diagram.

**JAPE** Java Annotation Patterns Engine.

**JVM** Java Virtual Machine.

**MBT** Model-Based Testing.

**MDE** Model-Driven Engineering.

**NARCIA** NAatural language Requirements Change Impact Analyzer.

**NL** Natural Language.

**NLP** Natural Language Processing.

**NPs** Noun Phrases.

**OMG** Object Management Group.

## *Acronyms*

---

**PP** Prepositional Phrase.

**RD** Requirement Diagram.

**RE** Requirements Engineering.

**REGICE** REquirements Glossary term Identification and ClustEring tool.

**RETA** REquirements Template Analyzer.

**RQ** Research Question.

**RSA** Rational Software Architect.

**SysML** Systems Modeling Language.

**TCC** Template Conformance Checking.

**TIM** Traceability Information Model.

**UML** Unified Modeling Language.

**VPs** Verb Phrases.

# Chapter 1

## Introduction

### 1.1 Context

Requirements Engineering (RE) is the systematic process of identifying, specifying, analyzing, and maintaining the requirements for a proposed system. Requirements capture the capabilities, characteristics, qualities, and operational constraints of a system envisaged by the users or other stakeholders of a system [Sommerville and Sawyer, 1997, Young, 2004, van Lamsweerde, 2009, Pohl, 2010, Hull et al., 2011, Pohl and Rupp, 2011, Chemuturi, 2012].

Broadly, requirements are specified using two main formats: natural language and formal language. Natural language (NL) is arguably the more common of the two in industrial settings. NL tends to be easier to understand by the stakeholders as no special training is required. NL further has the advantage that it can be used in any problem domain and is flexible to accommodate arbitrary abstractions and refinements [Pohl and Rupp, 2011]. Despite these advantages, NL is prone to ambiguity, vagueness, incompleteness, and inconsistency [Pohl and Rupp, 2011]. Unless appropriate analysis, verification and validation processes are put in place, the use of NL can negatively impact the quality of requirements specifications.

This dissertation aims to capitalize on the benefits of NL for specifying requirements, and yet provide automation for a number of complex and laborious quality assurance and analysis tasks that need to be performed over NL requirements. The main enabling technology in this dissertation is Natural Language Processing (NLP). By providing the ability to identify and extract semantic information from NL requirements, NLP plays a vital role in automated requirements analysis.

The majority of the research presented in this dissertation has been done in collaboration with SES [SES, 2016], a leading satellite solutions provider worldwide. The software solutions developed by SES require them to follow development standards set by the European Space Agency (ESA). A typical project at SES involves multiple stakeholders, with high-level requirements that originate from numerous sources. The requirements, which are by and large written in NL, constitute the basis for contractual agreements between the stakeholders and thus correlate directly with cost, responsibility and legal liability considerations. Due to these factors, SES has a strong interest in NL requirements analysis solutions that can provide assistance in better manipulation and quality assurance of requirements. We have developed in collaboration with SES several solutions aimed at automated

requirements analysis. While informed by practical considerations at SES, our solutions are not restricted to SES and generalize to other contexts in which NL requirements are used. One of the research solutions developed in this thesis –the subject of Chapter 7– is the result of a collaboration with Delphi Automotive Systems Luxembourg [Delphi, 2016]. Delphi is a leading parts supplier in the automotive industry. Similar to the solutions built in collaboration with SES, the results of our collaboration with Delphi are generalizable to a wider variety of contexts.

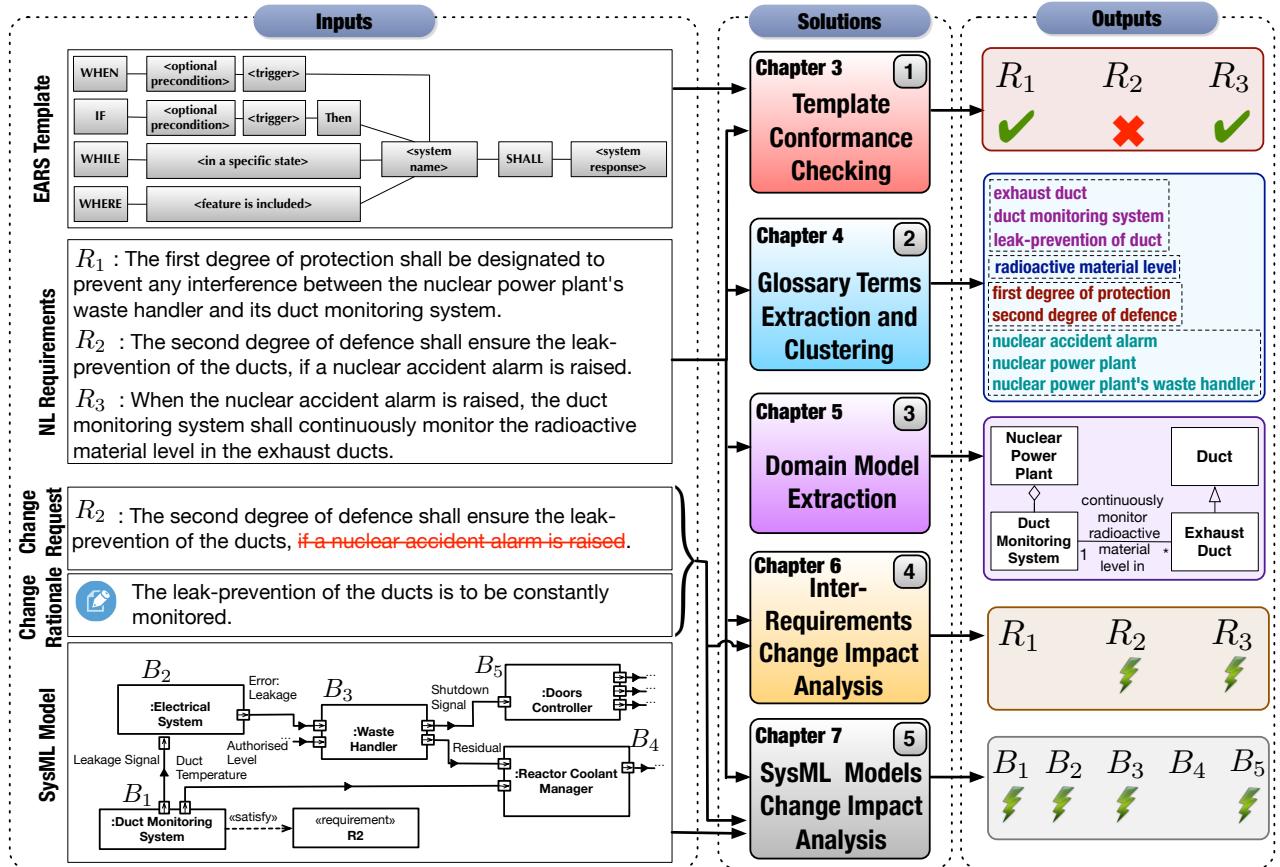


Figure 1.1. Dissertation overview and organization.

## 1.2 Contributions and Organization

In this dissertation, we investigate selected facets of requirements analysis in an industrial context and devise, using NLP, novel solutions for automating the analysis tasks under investigation. Figure 1.1 provides an overview of the different solutions developed throughout this dissertation and illustrates the solutions over a small sample of NL requirements derived from the Finnish nuclear plant safety regulations [STUK, 2016]. The requirements have been slightly modified from their original form to better exemplify our solutions.

Specifically, the solutions developed in this dissertation are as follows:

- Template Conformance Checking (TCC), marked (1) in Figure 1.1, automatically validates the application of a requirements template on requirements sentences. This solution requires as

input a set of NL requirements and the intended requirements template to conform with. For the example requirements of Figure 1.1, the intended template is Easy Approach to Requirements Syntax (EARS) [Mavin et al., 2009, Mavin and Wilkinson, 2010, Gregory, 2011]. This solution returns conformance diagnostics for the input requirements and, when a requirement is non-conformant, the reason for non-conformance. For the example requirements in the figure,  $R1$  and  $R3$  are conformant to the EARS template, whereas  $R2$  is not. The reason why  $R2$  is non-conformant is that the conditional phrase “if a nuclear accident alarm is raised” appears at the end of the requirements statement rather than at the beginning, as prescribed by the EARS template. It must be noted that among the five solutions depicted in Figure 1.1, only TCC is targeted at structured requirements that conform to a template; all other solutions are targeted at the more general case where the NL requirements are unrestricted, and thus do not necessarily follow a template. Our TCC solution has been published in a conference paper [Arora et al., 2013a], and a follow-up journal paper [Arora et al., 2015a]. An additional workshop paper [Arora et al., 2014b] and a tool paper [Arora et al., 2013b] have been published to provide insights into the technical details of our solution. We cover TCC in **Chapter 3**.

- Glossary Terms Extraction and Clustering (GE), marked (2) in Figure 1.1, automatically extracts potential glossary terms from an input set of requirements statements, and clusters the extracted terms based on the similarity between the terms. The example output from GE in Figure 1.1 shows four clusters of relevant terms for the input requirements ( $R1$ - $R3$ ). Such clusters help requirements analysts in selecting the most suitable glossary terms, writing definitions for these terms, and identifying variant phrases that may have been used for referring to the same concept. For instance, the cluster with keywords “first degree of defence” and “second degree of protection” highlights a potential variation (namely “defence” versus “protection”) in the terminology. Such variations, unless they are unified or made explicit, can result in inconsistencies. Our GE solution accounts for both syntactic and semantic similarities between terms when clustering them. We provide practical guidelines for selecting the optimal number of clusters for a given set of requirements. This work has been published in a conference paper [Arora et al., 2014a]. A follow-up journal paper has been written, which is currently undergoing a minor revision for publication at IEEE Transactions of Software Engineering. We cover GE in **Chapter 4**.
- Domain Model Extraction (ME), marked (3) in Figure 1.1, automatically extracts a domain model from an input set of NL requirements. Domain models provide a precise and yet intuitive way for domain experts to capture their otherwise tacit knowledge about the salient concepts in a domain and the relationships between these concepts [Schneider, 2009]. The ME solution is a generalization of our GE work, as the identification of the salient domain concepts is similar across both approaches. The relationships between these terms, otherwise implicitly captured by the clusters in the GE approach are explicated by our ME solution. The output of ME depicted in Figure 1.1 is a domain model fragment extracted automatically from our example requirements. Our ME solution is rule-based. We propose in our work new rules to improve the completeness of automated domain model extraction. Our ME solution has been published as a conference paper [Arora et al., 2016]. We cover ME in **Chapter 5**.
- The last two solutions, marked (4) and (5) in Figure 1.1, both have to do with requirements change analysis. Uncontrolled requirements evolution presents one of the biggest risks to the

success of software projects [van Lamsweerde, 2009]. Manually identifying the impact of requirements changes is an arduous task; automated assistance aimed at facilitating this task is therefore beneficial. We propose two change impact analysis solutions addressing two different dimensions of requirements change propagation: propagation of change from one requirement to another (inter-requirements), and propagation of change from requirements to design (requirements-to-design):

- Our Inter-Requirements Change Impact Analysis (IR-CIA) solution, marked (4) in Figure 1.1, automatically analyzes the impact of changing one requirement on other requirements. In addition to a set of requirements, IR-CIA requires two additional inputs: the change request and the change rationale. In Figure 1.1, the change request is the deletion of the condition in  $R_2$ , with the rationale being that leak-prevention should be monitored constantly, rather than only when a nuclear accident alarm has been raised. For a given change and a given rationale, our solution computes a quantitative score (not shown in Figure 1.1) signifying how likely it is for each requirements statement to be impacted. Our solution is equipped with guidelines to assist analysts in deciding which requirements are most likely to be impacted based on the scores and thus need to be manually inspected. For instance, the IR-CIA results in Figure 1.1 suggest that for the above-described change to  $R_2$ , only  $R_3$  is likely to be impacted but not  $R_1$ . Our IR-CIA solution, which we cover in **Chapter 6**, has been published as a conference paper [Arora et al., 2015a] and a tool paper [Arora et al., 2015c].
- Our Requirement-to-Design Change Impact Analysis (RD-CIA) solution, marked (5) in Figure 1.1, focuses on propagating changes from NL requirements to system designs, when the requirements and design elements are expressed using Systems Modeling Language (SysML) models. Our RD-CIA solution requires four inputs: the requirements, SysML design models with dependency links between requirements and design blocks, a change request, and the rationale for the change. The solution computes a quantitative score (not shown) signifying how likely it is for any given design element to be impacted by the requirements change in question. Guidelines similar to those in our IR-CIA solution are utilized for identifying design elements that are most likely to be impacted. For instance, in Figure 1.1, the RD-CIA output indicates that for the given change in  $R_2$ , all blocks except  $B_5$  are likely to be impacted. Our RD-CIA solution has been published as a conference paper [Nejati et al., 2016] and is covered in **Chapter 7**.

Before proceeding to present the dissertation contributions in Chapters 3 through 7, we provide in **Chapter 2** the overall background for our work.

# Chapter 2

## Background

This chapter provides background information for the dissertation. The content of the chapter is organized under four headings: (1) Natural Language (NL) requirements, (2) Natural Language Processing (NLP), (3) Unified Modeling Language (UML) and Systems Modeling Language (SysML), and (4) Quality Attributes.

### 2.1 Natural Language Requirements

NL requirements may come in many forms, including (IEEE-830 style) “shall” requirements, use cases, user stories, scenarios, and feature lists [Pohl and Rupp, 2011]. Some of these representations are more suitable for specifying certain requirements types than others. For example, use case descriptions and user stories are more suited for specifying user requirements (problem domain) than system requirements (solution domain). A brief description of some of the most common NL requirements representations follows:

**IEEE-830 style “shall” requirements** [IEEE Computer Society and Board, 1998] are expressed as statements starting with “The system shall ...”, with the possibility to replace “system” with an appropriate system or actor name. For instance, the statement “The temperature sensor shall send the temperature of the ducts to the duct monitoring system” is a shall requirement. The modal verb “shall” in the requirements statement can be replaced by other close variants, such as “will” and “should”. The modal verb signifies how binding a requirement is. Requirements with “shall” are typically taken to be legally binding; those with “should” are not legally binding; and those with “will” are meant for the future [Rupp, 2009]. A wide range of requirements types are expressible using the IEEE-830 format, which is not necessarily the case for other NL formats [IEEE Computer Society and Board, 1998]. Additionally, the IEEE-830 format requires virtually no prior training and is easily understandable by all stakeholders.

**Use cases** are generalized descriptions of interactions between a system-to-be and the relevant actors. Use cases are used primarily for capturing functional requirements, and more specifically the system’s functional behavior upon a request from an actor under specific conditions. User requirements, e.g., user goals and the tasks they should be able to perform, are also captured in use cases [Wiegers, 2005]. A use case description typically includes at least the following information:

name of the use case, brief description, precondition(s), actors, dependencies, flow of events, basic and alternative flows, and postcondition(s).

**User stories** are brief descriptions of the functionality of a system described from a users' perspective. User stories are most commonly used in agile software development and are generally specified using a pattern - *As a < type of user >, I want < some goal > so that < some reason >*. An example instantiation of this pattern would be "As a system administrator, I want to generate a report of all the sessions in the archived sessions list with a timestamp so that a daily log of handled sessions is maintained".

**Feature list** is yet another NL requirements format commonly used in agile settings [Chemuturi, 2012]. Feature lists specify the features that need to be built into a system from a client / user's perspective. Feature lists are used in an iterative fashion and are usually reset after a fixed time frame, e.g., two weeks. The list and the description of the features are specified along with any supplementary information about the features such as their priority, and the current version of the feature implemented on the system. Figure 2.1 shows an example features list, with two features for managing the comments of users in an application.

**Table 2.1.** Feature list example.

<b>Id</b>	<b>Feature</b>	<b>Description</b>	<b>Priority</b>	<b>Development Effort (Man Days)</b>
F001	Editable Comments Section	Currently, users are only allowed to add new comments. Amend it to allow users to edit their existing comments.	Should	3
F002	Deletion Warning	A pop-up appears asking the users to confirm they want to delete their comments and remind them that comments will be irretrievable after deletion.	Must	1

All the solutions in this dissertation are grounded on IEEE-830 style "shall" requirements. This was necessitated mainly by the need to align the developed solutions with the development practices at the collaborating companies. In the remainder of the dissertation, when referring to an NL requirement or requirements statement, the IEEE-830 format is implied, unless explicitly stated otherwise.

## 2.2 Natural Language Processing

Natural Language Processing (NLP) refers to the computerized understanding, analysis, manipulation, and generation of natural language [Manning and Schütze, 1999, Jurafsky and Martin, 2009]. Table 2.2 shows the NLP techniques used in this dissertation. A general overview of these techniques is provided below. More details are provided in the individual chapters based on the context and needs. For the purpose of illustration, let  $R_x$  denote the following requirements statement in the rest of this section: "The system shall send the log messages to the database via the monitoring interface".

**Text chunking** is an NLP technique for delineating non-overlapping phrases in sentences (in our case requirements statements) [Jurafsky and Martin, 2009]. These phrases include, among others, Noun Phrases (NPs) and Verb Phrases (VPs). The NPs in  $R_x$  are "the system", "the log messages",

**Table 2.2.** NLP techniques applied for different approaches in this dissertation.

Chapter	Task	NLP Technique(s) Applied
Chapter - 3	Template Conformance Checking	<ul style="list-style-type: none"> <li>Text Chunking</li> <li>Phrase Structure Parsing</li> </ul>
Chapter - 4	Glossary Terms Extraction and Clustering	<ul style="list-style-type: none"> <li>Text Chunking</li> <li>Syntactic and Semantic Similarity Calculation</li> </ul>
Chapter - 5	Domain Model Extraction	<ul style="list-style-type: none"> <li>Text Chunking</li> <li>Phrase Structure Parsing</li> <li>Dependency Parsing</li> </ul>
Chapter - 6	Inter-Requirements Change Impact Analysis	<ul style="list-style-type: none"> <li>Text Chunking</li> <li>Syntactic and Semantic Similarity Calculation</li> </ul>
Chapter - 7	SysML Models Change Impact Analysis	<ul style="list-style-type: none"> <li>Text Chunking</li> <li>Syntactic and Semantic Similarity Calculation</li> </ul>

“the database”, and “the monitoring interface”. The VP in  $R_x$  is “shall send”. Text chunking is discussed in more detail in Chapter 3.

**Phrase structure parsing** is an NLP technique for determining the hierarchical phrasal structure of sentences. Whereas text chunking returns a flat structure, phrase structure parsing returns a (hierarchical) tree. Using this tree, the constituent phrases of a sentence can be retrieved at the desired level of abstraction. Stated more precisely, phrase structure parsing returns both atomic phrases, i.e., phrases that can not be split further, and compound phrases, i.e., phrases that can be further split into subphrases. For example, in  $R_x$ , the atomic NPs are the same as the ones identified by text chunking. Phrase structure parsing further identifies a compound NP, namely “the database via the monitoring interface”. Phrase structure parsing is discussed in more detail in Chapters 3 and 5.

**Dependency parsing** is an NLP technique for extracting the grammatical structure of sentences. More specifically, dependency parsing identifies the dependency relationships between the words in a sentence. For example, in  $R_x$ , one of the many dependencies returned by a dependency parser is  $nsubj(send-4, system-2)$ , stating that “system” is a subject of the verb “send”. The numbers in this dependency correspond to the word sequence numbers in  $R_x$ . Dependency parsing is discussed in more detail in Chapter 5.

**Similarity calculation** is an NLP technique for computing a degree of similarity between different textual elements. The similarity is computed based on a measure, which may be a syntactic or a semantic one. Syntactic similarity signifies the similarity between the string content of different textual elements. For example, Dice’s coefficient [Dice, 1945] is a syntactic similarity measure that computes the similarity between two strings as twice the ratio of the number of common words between the strings to the total number of words in the strings. Dice’s coefficient between “nuclear power plant” and “nuclear accident”, for instance, would be 0.4. Semantic similarity measures compute the similarity of words based on their semantics or meaning in a dictionary. For example, the PATH measure computes the similarity between words based on the distance between them in a the-

saurus Is-a hierarchy graph. The PATH similarity score between “car” and “bicycle” is, for instance, 0.33, as, in an Is-a tree, both concepts are descendants of the concept of “wheeled vehicle”. Similarity measures are discussed in more detail in Chapters 4, 6, and 7.

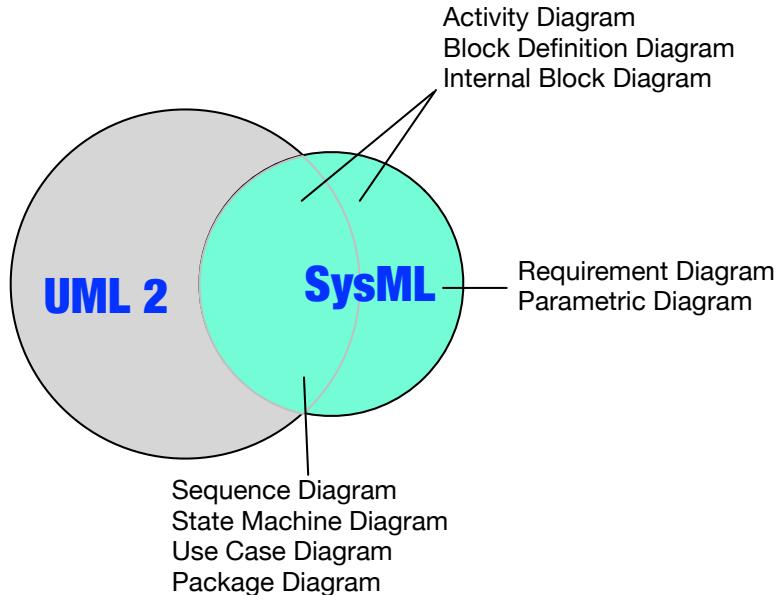


Figure 2.1. Relation between SysML and UML [OMG, 2016].

## 2.3 UML and SysML

The Unified Modeling Language (UML) is a general-purpose modeling language for specifying, visualizing, constructing, and documenting the artifacts of object-oriented software systems [Larman, 2005]. UML diagrams can be classified into two categories: structural and behavioral. Structural diagrams model the elements present in a software system and behavioral diagrams describe the system’s functionality.

The Systems Modeling Language (SysML) is a graphical modeling language for systems engineering that can be used for specification, analysis, design and verification of complex systems containing hardware, software, personnel, facilities and procedures [INCOSE, 2016]. SysML extensively reuses a subset of UML 2 and provides extensions to the existing diagrams in UML 2. Figure 2.1 shows the relation between SysML and UML 2. SysML reuses Sequence Diagrams, State Machine Diagrams, Use Case Diagrams, and Package Diagrams as-is from UML 2. Activity Diagrams, Block Definition Diagrams, and Internal Block Diagrams are the modified versions of existing UML 2 diagrams. Further, SysML introduces two new diagrams, i.e., the Requirement Diagram and the Parametric Diagram.

Chapters 5 and 7 of this dissertation use UML and SysML, respectively. Specifically, UML Class Diagrams are used in Chapter 5 for representing the domain models that we extract from NL requirements; SysML Requirement Diagrams, Internal Block Diagrams and Activity Diagrams are used in Chapter 7 for representing requirements, system design and the links between the two. A short introduction to these diagram types follows below. For more information about UML and SysML,

the reader can consult standard textbooks, e.g., [Larman, 2005, Stevens, 2006, Friedenthal et al., 2008, Holt and Perry, 2008]

**Class Diagrams** provide a structural view of an (object-oriented) software system. In particular, Class Diagrams visualize a system's classes, objects, interfaces, attributes and operations, and the relationships between the classes. Class diagrams are also commonly used for representing domain models, although there may be no immediate link between the domain and the software classes and responsibilities [Larman, 2005].

**Requirement Diagrams** visualize the requirements, both functional and non-functional, that the system must satisfy. The individual requirements are written in NL and mapped to a unique identifier (id). Requirements Diagrams also provide means for capturing the relationships between requirements and from requirements to other artifacts such as design models. The standard relationship types in SysML are contain, satisfy, derive, trace, verify, refine, and copy. Figure 2.3 provides a brief description of these relationship types. The Requirement Diagram type is new in SysML and without a precedent in UML 2. This diagram type was introduced due to the inability of a single UML diagram type to represent all classes of requirements, especially non-functional ones [Martorell et al., 2014].

**Table 2.3.** SysML Requirement Diagram relationship types.

Relationship Type	Description
Containment	Depicts a requirement split into multiple simpler requirements, with the contained requirements not adding or deleting any additional information to the original requirement.
Satisfy	Depicts a link between a requirement and a SysML element, where the design model element satisfies the requirement.
Derive	Depicts a link between a requirement derived from another requirement. Generally correspond to the requirements at the different levels of system hierarchy.
Trace	Depicts a trace link between a requirement and SysML elements .
Verify	Depicts a link between a test case and a requirement to show the possibility of requirement verification by a SysML element.
Refine	Depicts a link between a requirement and other SysML elements, such as, an activity diagram that further refine the NL requirement.
Copy	Depicts a link between a requirement and its exact copy. It is primarily useful for requirement reuse across product families and projects.

**Internal Block Diagrams** describe the internal structure of a block. Internal Block Diagrams further contain the ports (the interfaces of a block) and the connectors (the connections between the ports of blocks). The ports capture the information about the offered and required services of a block. The connectors describe the communication between blocks, signifying the flow of physical items or information between the blocks. Internal Block Diagrams are based on the UML Composite Structure Diagrams.

**Activity Diagrams** model the behavior of a system or a component by showing the workflows of activities in the system or component. Activity Diagrams can model the flow of data, control and even physical items (e.g., water). Activities can be further decomposed into sub-activities. Atomic activities are called actions. SysML Activity Diagrams are derived from UML Activity Diagrams.

## 2.4 Quality Attributes

The evaluations we perform over our proposed solutions aim at demonstrating that the solutions meet certain quality attributes. Below, we provide general definitions for the two main attributes considered in this dissertation: accuracy and usefulness. The exact meaning and the metrics we use to measure these attributes differ across the solutions, and are elaborated alongside the individual solutions described in the subsequent chapters.

- **Accuracy** refers to how close the actual results produced by a solution are to the desired results.
- **Usefulness** refers to the extent to which a solution helps practitioners for performing a certain task.

# Chapter 3

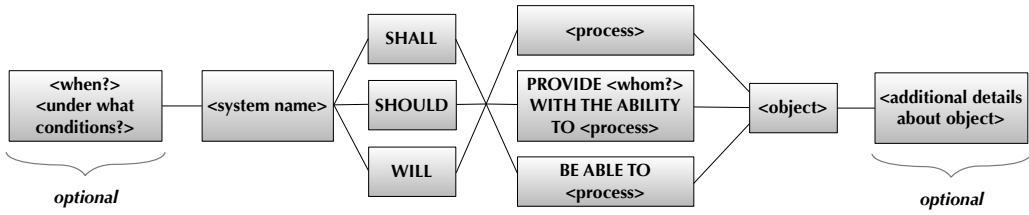
## Requirements Template Conformance Checking

Templates, also known as boilerplates, molds, or patterns [Pohl, 2010, Berry et al., 2003] are effective tools for increasing the precision of Natural Language (NL) requirements, for avoiding ambiguities that may arise from the use of unrestricted NL, and for making requirements more amenable to automated analysis [Withall, 2007, Pohl and Rupp, 2011, Uusitalo et al., 2011]. When templates are applied, it is important to verify that the requirements are indeed written according to the templates. If done manually, checking conformance to templates is laborious, presenting a particular challenge when the task has to be repeated multiple times in response to changes in the requirements.

In this chapter, using techniques from Natural Language Processing (NLP), we develop an automated approach for checking conformance to templates. Specifically, we present a generalizable method for casting templates into NLP pattern matchers and reflect on our practical experience implementing automated checkers for two well-known templates in the Requirements Engineering community, Rupp’s template [Pohl and Rupp, 2011] and Easy Approach to Requirements Syntax (EARS) template [Mavin et al., 2009, Mavin and Wilkinson, 2010, Gregory, 2011].

We have implemented our approach in a tool named REquirements Template Analyzer (RETA). We discuss the evaluation of our approach on four industrial case studies. Our results indicate that: (1) our approach provides a robust and accurate basis for checking conformance to templates; and (2) the effectiveness of our approach is not compromised even when the requirements glossary terms are unknown. This makes our work particularly relevant to practice, as many industrial requirements documents have incomplete glossaries.

**Structure.** The remainder of the chapter is structured as follows: Section 3.1 motivates and formulates the problem we aim to address in this chapter. It further outlines the research contributions of this chapter. Section 3.2 provides background information on requirements templates and NLP to the extent needed in our approach. Section 3.3 describes our approach for automation of template conformance checking. Section 3.4 discusses tool support. Sections 3.5 presents the case studies we have conducted to evaluate our approach. Section 3.6 identifies the limitations and analyzes threats to validity. Section 3.7 compares our approach with related work. Section 3.8 concludes the chapter with a summary and directions for future work.



**Figure 3.1.** Rupp's template [Pohl and Rupp, 2011].

### 3.1 Motivation and Contributions

Templates organize the syntactic structure of a requirements statement into a number of pre-defined slots. Figure 3.1 shows one of the well-known requirements templates, due to Rupp [Pohl and Rupp, 2011]. The template envisages six slots: (1) an optional condition at the beginning; (2) the system name; (3) a modal (shall/should/will) specifying how important the requirement is; (4) the required processing functionality; this slot can admit three different forms based on the manner in which the functionality is to be rendered (explained later in Section 3.2.1); (5) the object for which the functionality is needed; and (6) optional additional details about the object.

We show in Figure 3.2 three example requirements:  $R_1$  and  $R_2$  conform to Rupp's template; whereas  $R_3$  does not. For  $R_1$  and  $R_2$ , we show the different sentence segments and how they correspond to the Rupp's template slots. The fixed elements of the template are written in capital letters.

*R<sub>1</sub>:* The Surveillance and Tracking module SHALL PROVIDE the system administrator WITH THE ABILITY TO monitor system configuration changes posted to the database.

*R<sub>2</sub>:* AS SOON AS a power outage is detected, the Surveillance and Tracking module SHALL record a warning in the system alert log file.

*R<sub>3</sub>:* For each communication channel type, the system needs to maintain a configurable timeout parameter.

**Figure 3.2.** Example requirements:  $R_1$  and  $R_2$  conform to Rupp's template but  $R_3$  does not. The fixed elements of the template are written in capital letters.

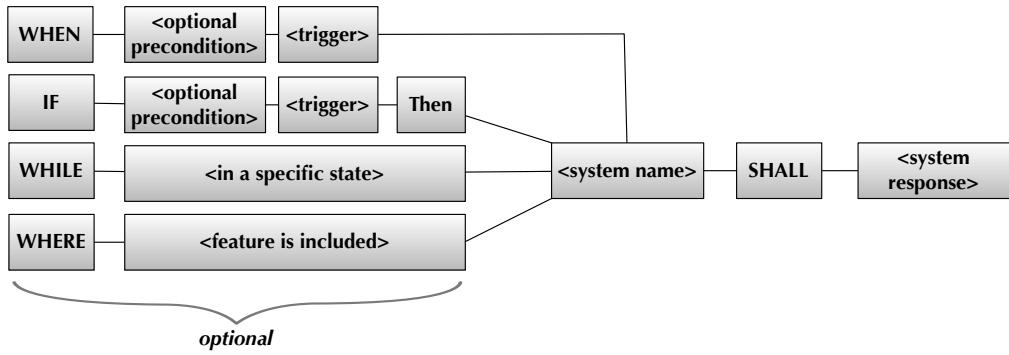
When templates are used, an important quality assurance task is to verify that the requirements conform to the templates. If done manually, this task can be time-consuming and tedious [Pohl and Rupp, 2011]. Particularly, the task presents a challenge when it has to be repeated multiple times in response to changes in the requirements. When the requirements glossary terms (domain keywords) are known, checking conformance to templates can be automated with relative ease. For example, consider  $R_1$  in Figure 3.2 and assume that “Surveillance and Tracking module” (system component), “system administrator” (agent), “monitor” (domain verb), and “system configuration change” (event) have been already declared as glossary terms. In this situation, an automated tool can verify conformance by checking that  $R_1$  is composed of a subset of the glossary terms (or terms with close syntactic resemblance to the glossary terms) and a subset of fixed template elements, put together in a sequence that is admissible by the template. The fixed elements may be leveraged for distinguishing different template slots. For example, whatever appears between `PROVIDE` and `WITH THE ABILITY TO` in  $R_1$  has to correspond to the  $\langle \text{whom?} \rangle$  (sub)slot.

This approach does not work when the glossary terms are unknown, because it can no longer distinguish sentence segments that correspond to the template slots. For example, in Rupp’s template, `<process>`, `<object>`, and `<additional details>` come in a sequence without any fixed elements in between. For an automated tool to deem  $R_1$  as conformant, it has to correctly distinguish these three slots in the following: “monitor system configuration changes posted to the database”. A second important issue is that even when a slot falls in between fixed elements, e.g., `<whom?>`, and is thus easy to delineate, there is no way to distinguish an acceptable filler for the slot from an unacceptable one, e.g., a grammatically-incorrect phrase. For instance, in  $R_1$ , we may accept “system administrator” as correctly filling `<whom?>`, but we may be unwilling to accept a grammatically-incorrect phrase such as “system administer” for the slot.

While building a glossary is an essential activity in any requirements project, the glossary is not necessarily available at the time one wants to check conformance to templates. In fact, based on our experience [Arora et al., 2014b], practitioners tend to identify and define the glossary terms *after* the requirements have sufficiently matured and are thus less likely to change. This strategy avoids wasted effort at the glossary construction stage, but it also means that the glossary will not be ready in time to support activities such as template conformance checking, which often take place *during* requirements writing. Furthermore, the literature suggests that glossaries may remain incomplete throughout the entire development process [Zou et al., 2010], thus providing only partial coverage of the glossary terms. This implies that automated techniques for checking conformance to templates would have limited effectiveness if such techniques rely heavily on the glossary terms being known *a priori*.

This chapter is motivated by the need to provide an automated and generalizable solution for Template Conformance Checking (hereafter, TCC) *without* reliance on a glossary. To this end, we make the following three contributions:

- We propose an approach for the automation of TCC using NLP. The main enabling NLP technology used in our approach is *text chunking*, which identifies sentence segments (chunks) without performing expensive analysis over the chunks’ internal structure, roles, or relationships [Jurafsky and Martin, 2009]. These chunks, most notably Noun Phrases (NPs) and Verb Phrases (VPs), provide a suitable level of abstraction over NL for characterizing template slots and performing TCC (Section 3.3). Our approach further utilizes NLP parsing when text chunking alone cannot conclusively determine template compliance, e.g., when requirements analysts elect to fill the template slots with complex NPs that include VPs in their makeup.
- We report on four case studies conducted in order to evaluate our approach. Two of these studies involve Rupp’s template, discussed above, and the other two involve another well-known template, called EARS, discussed further in Section 3.2.1. In both of the studies that use Rupp’s template, the requirements were written directly by professionals. As for the two studies using EARS, one – which is the largest case study reported in this chapter – was written directly by professionals as well, while the remaining study uses transformed requirements from one of our two case studies based on Rupp’s template. The results from our case studies indicate firstly, that our approach provides an accurate basis for TCC; and secondly, that the effectiveness of our approach is not compromised even when the glossary terms are unspecified.
- We provide tool support for TCC. Our tool, named RETA enables analysts to automatically check conformance to both Rupp’s and the EARS templates, and to obtain diagnostics about potentially problematic syntactic constructs in requirements statements.



**Figure 3.3.** The EARS template [Mavin et al., 2009].

## 3.2 Background

### 3.2.1 Requirements Templates

When properly followed, templates serve as a simple and yet effective tool for increasing the quality of requirements by avoiding complex structures, ambiguity, and inconsistency in requirements. Templates further facilitate automated analysis by making NL requirements more easily transformable into analyzable artifacts, e.g., models [Pohl and Rupp, 2011].

Several templates have been proposed in the Requirements Engineering literature. While our approach can be tailored to work with a variety of existing templates, we ground our work in this chapter on two templates, namely Rupp's and the EARS templates [Pohl and Rupp, 2011, Mavin et al., 2009]. Our choice is motivated by the reported use of these templates in the industry [Rupp, 2009, Gregory, 2011, Terzakis, 2013] and the availability of practitioner guidelines for the templates, e.g., [Pohl and Rupp, 2011, Mavin, 2012]. These guidelines present an advantage for training purposes and introducing templates in production environments.

**Rupp's template**, shown in Figure 3.1, was already introduced in Section 3.1 except for the different forms that the processing functionality slot (i.e., the template's fourth slot) can assume. Rupp's template distinguishes three types of processing functionality:

- *Autonomous requirements*, captured using the “⟨process⟩” form, state functionality that the system offers independently of interactions with users.
- *User interaction requirements*, captured using the “PROVIDE ⟨whom?⟩ WITH THE ABILITY TO ⟨process⟩” form, state functionality that the system provides to specific users.
- *Interface requirements*, captured using the “BE ABLE TO ⟨process⟩” form, state functionality that the system performs to react to trigger events from other systems.

**The EARS template**, shown in Figure 3.3, is made up of four slots: (1) an optional condition at the beginning; (2) the system name; (3) a modal; and (4) the system response depicting the behavior of the system. EARS distinguishes five requirements types using five alternative structures for its first slot [Mavin et al., 2009]:

- *Ubiquitous requirements* have no pre-condition, and are used for requirements that are always active.

- *Event-driven requirements* begin with `WHEN` and are used for requirements that are initiated by a trigger event.
- *Unwanted behavior requirements* begin with `IF` followed by `THEN` before the `<system name>`. These requirements are usually used for expressing undesirable situations.
- *State-driven requirements* begin with `WHILE` and are used for requirements that are active in a definite state.
- *Optional feature requirements* begin with `WHERE` and are used for requirements that need to be fulfilled when certain optional features are present.
- *Complex requirements* use a combination of above patterns, i.e., more than one condition type.

Compared to Rupp’s, the EARS template offers more advanced features for specifying conditions. In contrast, Rupp’s template enforces more structure than EARS over the non-conditional parts of requirements sentences, particularly by requiring an object (the `<object>` slot) to always be present.

### 3.2.2 Text Chunking

Text chunking is the core NLP technology underlying our approach. As previously defined in Section 2.2, text chunking is the process of decomposing a sentence into non-overlapping segments [Jurafsky and Martin, 2009]. The main segments of interest are Noun Phrases and Verb Phrases. A *Noun Phrase (NP)* is a segment that can be the subject or object of a verb. A *Verb Phrase (VP)*, sometimes also called a verb group, is a segment that contains a verb with any associated modal, auxiliary, and modifier (often an adverb). For example, a correct chunking of the requirements statement  $R$  in Figure 3.4(a) would yield the segments shown in Figure 3.4(b). As seen from this figure, segments generated by text chunking have a flat structure. This is in contrast to segments in a parse tree – generated by a natural language parser such as the Stanford Parser [Klein and Manning, 2016] – which can have arbitrary depths, as shown in Figure 3.4(c).

When a parse tree is not required for analysis, chunking offers two major advantages over parsing [Bird et al., 2009]: First, chunking is computationally less expensive, having a complexity of  $O(n)$ , with  $n$  denoting the length of a sentence, versus  $O(n^3)$  for (rule-based) parsing. This lower complexity makes text chunking more scalable. Scalability is an important consideration in our context where we need to deal with large requirements documents. Second, text chunking is more robust than parsing [Song et al., 2006], in the sense that it produces results in a large majority of cases; whereas parsing may fail to generate a parse tree, particularly when faced with unfamiliar input. Similar to scalability, robustness is an important consideration in our context, noting that requirements documents are highly technical and can deviate from commonly-used texts for training parsers, e.g., news articles.

While the above considerations make text chunking better suited to our context, the lack of a parse tree means that text chunking cannot reveal the complete semantics of requirements statements. In particular, text chunking cannot identify complex phrases, and would instead find only the atomic phrases that make up complex ones. For example, the sentence segment “The information technology tools used in the design of systems performing safety functions” in the example of Figure 3.4 is a complex noun phrase. The chunks in Figure 3.4(b) capture only the atomic phrases of this complex noun phrase; whereas, the nodes in the parse tree of Figure 3.4(c) further capture the complex noun phrase in its entirety, as marked on the figure. We discuss the implications of complex phrases for TCC in Section 3.3.3. Below, we describe how a text chunker is implemented in an NLP environment.

*R* : The information technology tools used in the design of systems performing safety functions shall be assessed for safety implications on the end-product.

(a)

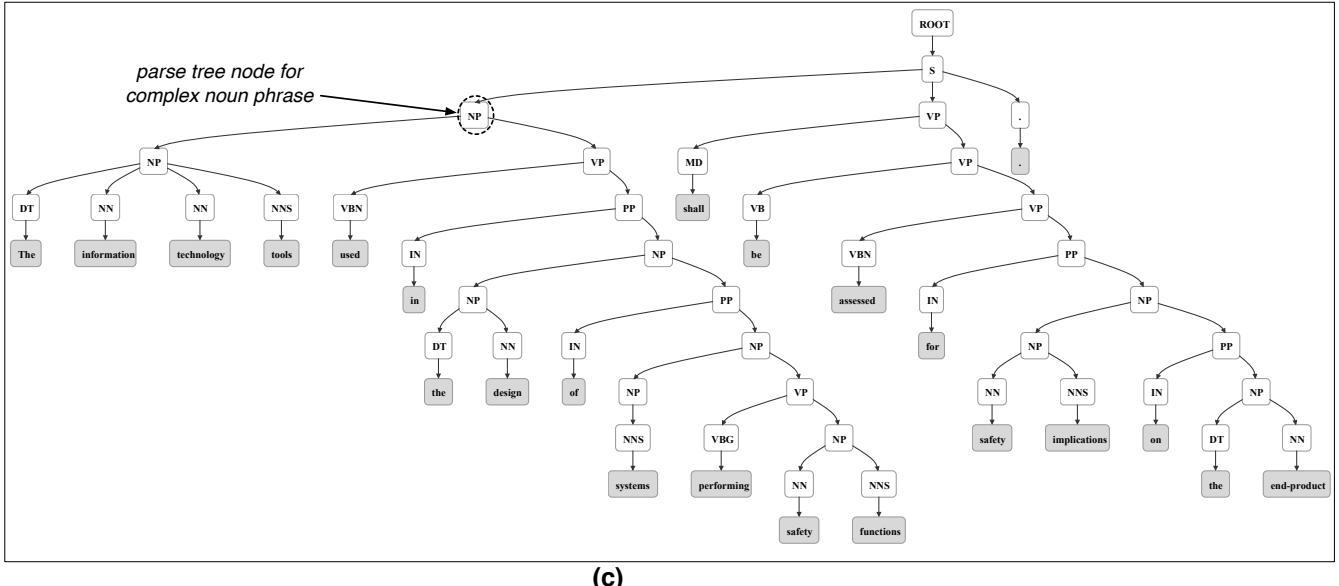
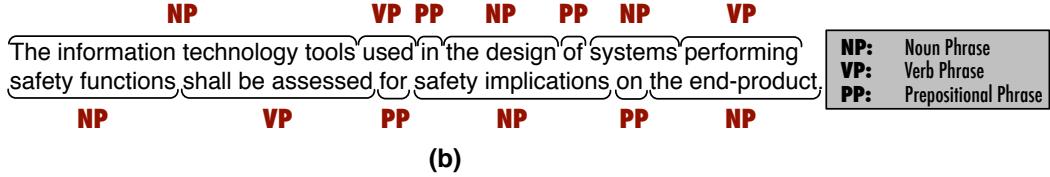


Figure 3.4. (a) An example requirements statement, (b) its sentence chunks, and (c) its full parse tree.

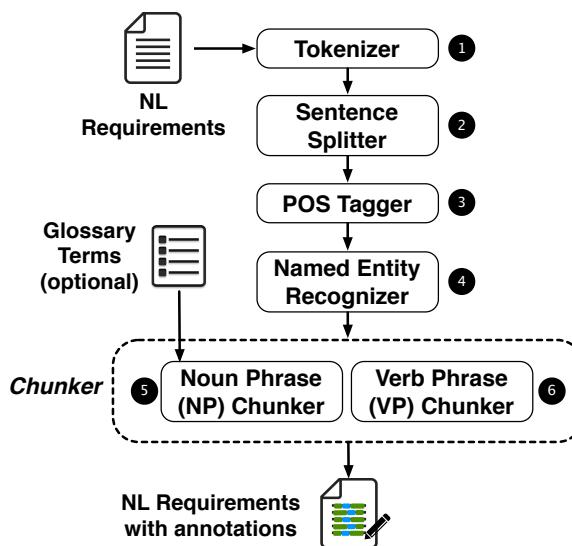


Figure 3.5. NLP pipeline for text chunking.

A text chunker is a pipeline of NLP modules running in a sequence over an input document. The (generic) pipeline for chunking is shown in Figure 3.5. As we explain in Section 3.5.4, this pipeline can be instantiated in many ways, as there are alternative implementations for each step in

the pipeline. The first module, the Tokenizer, breaks up the input into tokens. A token can be a word, a number or a symbol. The Sentence Splitter divides the text into sentences. The POS Tagger annotates each token with a part-of-speech tag. These tags include among others, adjective, adverb, noun, verb. Most POS Taggers use the Penn Treebank tagset [Marcus et al., 1993]. Next is the Name Entity Recognizer, where an attempt is made to identify named entities, e.g., organizations and locations. In a requirements document, the named entities can further include domain keywords and component names. The main and final step is the actual Chunker. Typically, but not always, NP and VP chunking are handled by separate modules, respectively tagging the noun phrases and verb phrases in the input. When a glossary of terms is available, one can instruct the NP Chunker to treat occurrences of the glossary terms in the input as named entities, thus reducing the likelihood of mistakes by the NP Chunker. To what extent the glossary is useful for TCC is the subject of RQ2 (see Section 3.5.6). Once processed by the pipeline of Figure 3.5, a document will have annotations for tokens, sentences, parts-of-speech, named entities, noun phrases, and verb phrases. We use these annotations for automating TCC, as we explain in Section 3.3.

### 3.2.3 Pattern Matching in NLP

As we elaborate in Section 3.3, we represent templates as BNF grammars. This representation enables the definition of pattern matching rules for checking template conformance. For implementing a BNF grammar over NL statements, we use JAPE (Java Annotation Patterns Engine). JAPE is a regular-expression-based pattern matching language, available as part of the GATE NLP workbench [GATE User Guide, 2016]. Figure 3.6 shows an example JAPE script, which checks conformance to Rupp's *Autonomous* requirements type.

```

1. Phase: MarkConformantSegment
2. Input: Condition NP VP Token
3. Options: control = appelt
4.
5. Rule: MarkConformantSegment_RuppAutonomous
6. (
7.   ((Condition)(NP):system_with_condition) |
8.   (NP): system_without_condition)
9.   (VP, VP.startsWithValidMD == "true",
10.    !VP contains {Token.string == "provide"}):process
11.   (NP):object
12. ):label
13. -->
14. :label.Conformant_Segment =
15. {explanation = "Matched pattern: Autonomous",
16. :system_with_condition.System_Name = {},
17. :system_without_condition.System_Name = {},
18. :process.Process = {},
19. :object.Object = {}}

```

**Figure 3.6.** JAPE script for Rupp's Autonomous type.

Each JAPE script consists of a set of phases, with each phase made up of a set of rules. In the script of Figure 3.6, we have a single phase, named *MarkConformantSegment* (line 1), which includes a single rule, named *MarkConformantSegment\_RuppAutonomous* (line 5). The phase could be extended with rules for checking conformance to Rupp's *User Interaction* and *Interface* requirements types. Each rule in JAPE consists of a left hand side (LHS) and a right hand side (RHS), which are

separated by `->` (line 13). The LHS specifies the annotation pattern that needs to be matched (lines 6–12), and the RHS – the action to be taken when a match is found (lines 14–19). The LHS in the rule of Figure 3.6 matches the pattern for Rupp’s *Autonomous* requirements type. The corresponding action in the RHS annotates as `Conformant_Segment` the entire segment matching the pattern (lines 14–15). The RHS has further actions for delineating `System_Name`, `Process`, and `Object` (lines 16–19). Note that `System_Name` needs to be recognized both in the presence and absence of a condition. Two temporary annotations, namely `system_with_condition` (line 7) and `system_without_condition` (line 8) have been defined in the LHS to enable detection of `System_Name` in both cases. The RHS of a JAPE rule can optionally contain Java code for manipulating the annotations. An example of an RHS with embedded Java code can be seen in Figure 3.10.

JAPE provides various options for controlling the results of annotations when multiple rules match the same section in text, or for controlling the text segment that is annotated on a match. These options are *brill*, *appelt*, *all*, *first*, and *once* [GATE User Guide, 2016]. In our work, we make use of *brill*, *appelt* and *first*. *Brill* means that when more than one rule matches the same region of text, all of the matching rules are fired, and the matching segment can have multiple annotations. This is useful, for example, while detecting potential ambiguities in a requirements sentence: if multiple ambiguities are present in a given text segment, all the ambiguities will be annotated. *Appelt* means that when multiple rules match the same region, the one with the maximum coverage (i.e., longest text segment) is fired. This can be used, for example, as shown in Figure 3.6, where we want to assign a unique type to a requirement that is most appropriate. *First* means that when there is more than one rule matching, the first one matching is fired without trying to get the longest sequence. This is useful, for example, when delineating sentences. To delineate a sentence, we need to match a sequence of tokens followed by a full stop. Using *first* enables us to correctly match individual sentences; whereas using *appelt* would result in matching the longest possible sequence, i.e., entire paragraphs.

## 3.3 Approach

In this section, we describe how to use the annotations produced by (an instantiation of) the text chunking pipeline in Figure 3.5 for automating TCC. Specifically, we show how Rupp’s and the EARS templates can be represented as BNF grammars over the annotations resulting from text chunking and then verified automatically in an NLP framework. Rupp’s and the EARS templates are the basis for the case studies in Section 3.5.

### 3.3.1 Expressing Templates as Grammars

In Figures 3.7(a) and 3.7(b), we provide the BNF grammars for Rupp’s and the EARS templates in terms of the annotations generated by the pipeline of Figure 3.5. For simplicity, in the grammars, we abstract from nested tags. For example, on line R.2, we use `<vp-starting-with-modal>` to denote a verb phrase (`<vp>`) that contains a modal at its starting offset. Similarly `<infinitive-vp>` denotes a `<vp>` starting with “to”.

In both templates, requirements can start with an optional condition. Rupp’s template does not provide any detailed syntax for the conditions, recommending only the use of the following conditional key-phrases: `IF` for logical conditions; and `AFTER`, `AS SOON AS`, and `AS LONG AS` for temporal conditions. EARS, in contrast, differentiates the types of requirements by the `<opt-condition>` rule

<pre> R.1. &lt;template-conformant&gt; ::=  R.2.   &lt;opt-condition&gt; &lt;np&gt; &lt;vp-starting-with-modal&gt; &lt;np&gt; R.3.     &lt;opt-details&gt;   R.4.   &lt;opt-condition&gt; &lt;np&gt; &lt;modal&gt; "PROVIDE" &lt;np&gt; R.5.     "WITH THE ABILITY" &lt;infinitive-vp&gt; &lt;np&gt; &lt;opt-details&gt;   R.6.   &lt;opt-condition&gt; &lt;np&gt; &lt;modal&gt; "BE ABLE" &lt;infinitive-vp&gt; R.7.     &lt;np&gt; &lt;opt-details&gt; R.8.   &lt;opt-condition&gt; ::= ""   R.9.     &lt;conditional-keyword&gt; &lt;token-sequence&gt; R.10.   &lt;opt-details&gt; ::= ""   R.11.     &lt;token-sequence-without-subordinate-conjunctions&gt; R.12.   &lt;modal&gt; ::= "SHALL"   "SHOULD"   "WILL" R.13.   &lt;conditional-keyword&gt; ::= "IF"   "AFTER"   "AS SOON AS"   R.14.     "AS LONG AS" </pre>	<pre> E.1. &lt;template-conformant&gt; ::=  E.2.   &lt;np&gt; &lt;vp-starting-with-modal&gt; &lt;system-response&gt;   E.3.   &lt;opt-condition&gt; &lt;np&gt; &lt;vp-starting-with-modal&gt; E.4.     &lt;system-response&gt; E.5.   &lt;opt-condition&gt; ::= ""   E.6.     "WHEN" &lt;opt-precondition&gt; &lt;token-sequence&gt;   E.7.     "IF" &lt;opt-precondition&gt; &lt;token-sequence&gt; "THEN"   E.8.     "WHILE" &lt;token-sequence&gt;   E.9.     "WHERE" &lt;token-sequence&gt;   E.10.   &lt;opt-condition&gt; E.11.   &lt;opt-precondition&gt; ::= ""   &lt;np&gt; &lt;vp&gt; (&lt;np&gt;)? E.12.   &lt;system-response&gt; ::=  E.13.     &lt;token-sequence-without-subordinate-conjunctions&gt; E.14.   &lt;modal&gt; ::= "SHALL" </pre>
---	---

(a)

(b)

Figure 3.7. BNF grammars for (a) Rupp’s template and (b) the EARS template.

in E.5 - E.10. E.10 captures the *complex* requirements type (Section 3.2.1) in EARS by the use of recursion in the  $\langle \text{opt-condition} \rangle$  slot.

A restriction that can be made in both templates is for the conditional segment to always end with a comma (”,”). This restriction may however be too constraining because commas can be easily forgotten or applied according to one’s personal preferences for punctuation. To avoid relying exclusively on the presence of a comma, one can employ heuristics for identifying the conditional segment in a requirement. In particular, one can use the system name (an NP) followed by a modal (e.g., SHALL) as an anchor for identifying the conditional part. For example, consider the following requirement  $R$  = “When a GSI component constraint changes STS SHALL deliver a warning message to the system operator”. In Rupp’s template, the heuristic capturing the syntax of  $R$  is  $\langle \text{conditional-keyword} \rangle \langle \text{sequence-of-tokens} \rangle \langle \text{np} \rangle \langle \text{vp-starting-with-modal} \rangle \langle \text{np} \rangle \langle \text{opt-details} \rangle$ .

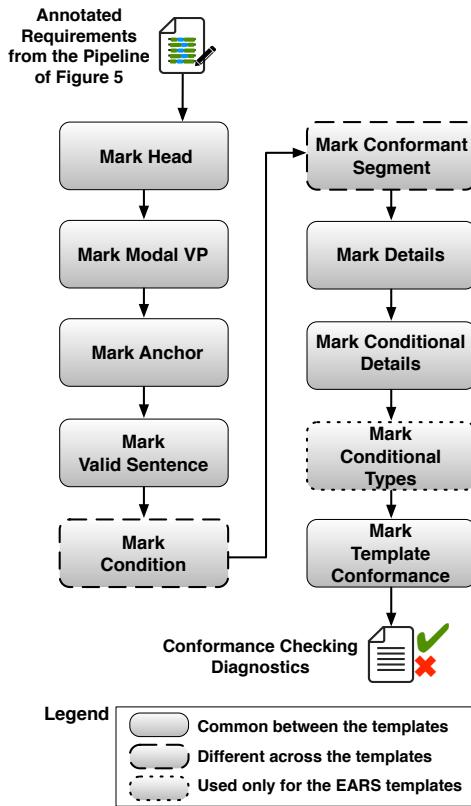
Template-specific keywords, e.g., the modals and the conditional keywords, are grouped into keyword lists, called gazetteers [GATE User Guide, 2016]. These lists decouple TCC rules from template-specific keywords, thus avoiding the need to change the rules when the keywords change.

Lastly, for the optional details in Rupp’s and the system response in the EARS template, we accept any sequence of tokens as long as the sequence does not include a subordinate conjunction (e.g., after, before, unless). The rationale here is that a subordinate conjunction is very likely to introduce additional conditions. Both Rupp’s and the EARS templates envisage that such conditions must appear at the beginning and not the end of requirements statements. Checking conformance to the rules in Figure 3.7 can be implemented using NLP pattern matching as we describe next.

### 3.3.2 Conformance Checking via Pattern Matching

TCC starts with the text chunking pipeline, shown and discussed in Section 3.2.2. Text chunking identifies tokens (along with their parts of speech), NPs, VPs, and named entities. Following text chunking, another text processing pipeline is executed. This second pipeline, shown in Figure 3.8, is composed of pattern matchers for recognizing template grammars. We use the JAPE language, introduced in Section 3.2.3, for implementing these pattern matchers.

Below, we outline each of the steps in the pipeline of Figure 3.8, showing the annotations generated by each step over the requirements of Figure 3.9. We further present the JAPE implementation of selected steps to illustrate the NLP machinery behind the approach:

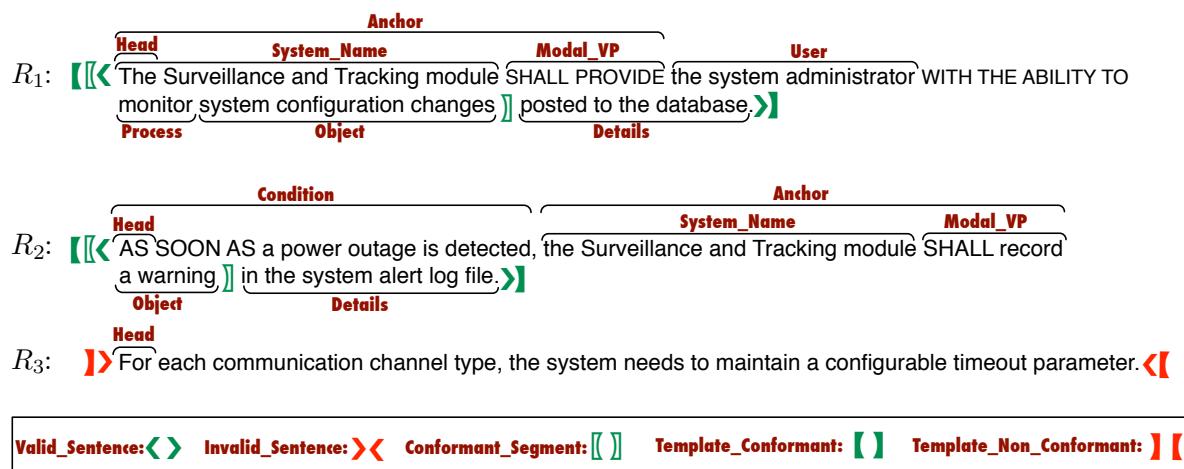


**Figure 3.8.** Pipeline for Template Conformance Checking.

1. **Mark Head** marks the starting word in a requirements sentence, denoted `Head` in the examples of Figure 3.9.
2. **Mark Modal VP** marks the VP that starts with a modal. A requirements statement typically has only one modal. If more than one modal is found, the first modal is annotated and a warning is generated to bring to the user's attention the presence of multiple modals. The annotation resulting from this step is denoted `Modal_VP` in Figure 3.9. Note that  $R_3$  contains no valid modal VP, hence the requirement is not having a `Modal_VP` annotation.
3. **Mark Anchor** first tags as `System_Name` the NP that precedes the modal. Subsequently, the step tags as `Anchor` the system name and the VP (including the modal) that follows `System_Name`. The anchor is later used for delineating the conditional slot. In the examples of Figure 3.9,  $R_3$  has no `Anchor` marked in it because the `Modal_VP` annotation is absent from this requirement.
4. **Mark Valid Sentence** marks as `Valid_Sentence` all sentences containing the anchor described above, subject to the constraint that the anchor is either at the beginning of a sentence or preceded by a segment starting with a conditional keyword. Sentences that do not have an anchor or fail to meet the additional constraint are marked as `Invalid_Sentence`. In the examples of Figure 3.9,  $R_1$  and  $R_2$  are valid sentences whereas  $R_3$  is an invalid one.
5. **Mark Condition** marks the optional condition in those valid sentences that start with a conditional keyword. The text between the beginning of such a sentence up to the anchor is marked as being a `Condition`. The scripts for marking conditions are different for Rupp's and EARS, as the syntactic structure of the conditions differs across these templates.
6. **Mark Conformant Segment** marks as `Conformant_Segment` the segment of a valid sentence that complies with the template. Since conformance rules are different across the templates (R.1–R.7 and E.1–E.4 in Rupp's and EARS, respectively), the scripts for conformance checking

are also different. Figure 3.6, discussed earlier, shows an excerpt of the JAPE script for marking conformant segments for the *Autonomous* requirements type in Rupp's template.

7. **Mark Details** annotates as `Details` both the optional details envisaged by Rupp's template as well as the system response envisaged by EARS. Naturally, this annotation is relevant for only requirements sentences that contain a `Conformant_Segment`. The `Details` annotation applies to  $R_1$  and  $R_2$ , as depicted in Figure 3.9. Figure 3.10 shows the JAPE script for generating the `Details` annotation. Lines 11–24 of the script are written directly in Java. The Java code computes the beginning and ending offsets for the annotation, that is, from the end of the `Conformant_Segment` to the end of the sentence.



**Figure 3.9.** Annotations generated by the pipeline of Figure 3.8 over the example requirements of Figure 3.2.

8. **Mark Conditional Details** checks the details for terms that may imply additional conditions, notably, subordinate conjunctions, e.g., “whenever”, and “once”. If such terms are detected in the details segment, the segment will be additionally marked as `Conditional_Details`. For example, if a phrase such as “whenever logging is enabled” is appended at the end of  $R_2$  in Figure 3.9, the requirement will be deemed as having conditional details. Both Rupp's and the EARS templates mandate that the conditional part should appear at the beginning of a requirement. Hence, conditional details should trigger non-conformance (see step 10). Figure 3.11 shows the JAPE script for marking conditional details. The script identifies any segment already marked as `Details` in which a conditional keyword, denoted `Conditional` is present. The `Conditional` annotation is produced via a customizable list (gazetteer) of such keywords.
9. **Mark Conditional Types**, exclusive to the EARS template, distinguishes the different sub-parts of the condition slot, e.g., trigger and specific states. The script subsequently infers the requirements type based on the type of the condition used. Figure 3.12 shows an excerpt of the script, concerned with marking the condition type for event-driven requirements. For example, suppose that the conditional keyword of  $R_2$  (Figure 3.9) was `WHEN` instead of `AS SOON AS`.  $R_2$  would then be deemed conformant to EARS with its conditional type being event-driven.
10. **Mark Template Conformance** marks as `Template_Conformant` any valid sentence that contains a conformant segment, excluding those that have conditional details (see step 8). Any requirement without a `Template_Conformant` segment will be marked as `Template_Non_Conformant`. For example,  $R_1$  and  $R_2$  in Figure 3.9 are deemed conformant and  $R_3$  is deemed non-conformant to Rupp's template.

```

1. Phase: DoMarkDetails
2. Input: Sentence Conformant_Segment
3. Options: control = appelt
4.
5. Rule: MarkTemplateDetails
6. (
7. {Sentence contains Conformant_Segment}
8. ):sentence
9. -->
10. {
11.   AnnotationSet sentenceAs =
12. (gate.AnnotationSet)bindings.get("sentence");
13.   AnnotationSet compliantAs =
14. inputAS.get("Conformant_Segment").getContained(
15.     sentenceAs.firstNode().getOffset(),
16.     sentenceAs.lastNode().getOffset());
17.   Node start = compliantAs.lastNode();
18.   Node end = sentenceAs.lastNode();
19.
20.   if (start == null || end == null)
21.     return;
22.
23.   FeatureMap features = Factory.newFeatureMap();
24.   outputAS.add(start, end, "Details", features);
25. }
```

**Figure 3.10.** JAPE script for marking details.

```

1. Phase: DoMarkConditionalDetails
2. Input: Conditional_Details
3. Options: control = appelt
4.
5. Rule: MarkConditionalDetails
6. (
7. {Details contains Conditional}
8. )
9. :label
10. -->
11. :label.Conditional_Details= {}
```

**Figure 3.11.** JAPE script for marking conditional details.

```

1. Phase: DoMarkConditionalType
2. Input: Condition
3. Options: control = appelt
4.
5. Rule: MarkEventConditions
6. Priority: 20
7. (
8.   (
9.     {Condition.string ==~ "[Ww]hen(.)+" }
10.   )
11. ):label
12. -->
13. :label.ConditionType = {Type="Event-driven"}
```

**Figure 3.12.** JAPE script for marking the event-driven conditional type in EARS.

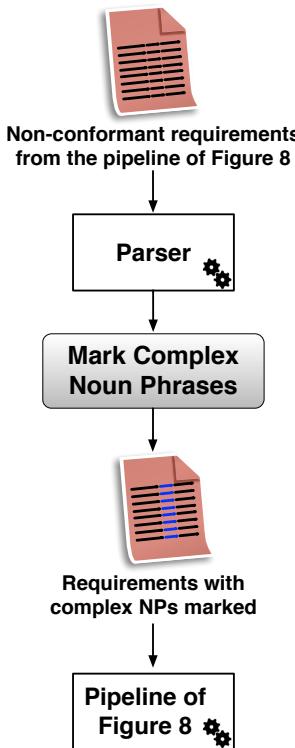


Figure 3.13. Enhancing TCC with parsing.

### 3.3.3 Handling Complex Phrases

As we discussed in Section 3.2.2, text chunking cannot identify complex phrases. To illustrate how this affects TCC, consider the requirements statement  $R$  in Figure 3.4(a). This statement conforms to the EARS template if we elect to fill the  $\langle$ system name $\rangle$  slot of the template with the complex noun phrase “The information technology tools used in the design of systems performing safety functions”. When a parse tree such as the one shown in Figure 3.4(c) is available, we can, as we explain below, deduce that the above phrase is indeed a noun phrase. Text chunking, in contrast, identifies only the atomic noun phrases of this complex noun phrase. Consequently, the pipeline of Figure 3.8 will mark  $R$  as non-conformant.

The absence of a parse tree seldom poses a problem for TCC. One of the main reasons why templates are used is to minimize the use of complex linguistic structures. Deeming as non-conformant a complex requirements statement such as that in the example of Figure 3.4(a) may be desirable as a way to bring the complexity to the attention of the analysts. If the analysts however decide that such complexity does not warrant further investigation, they will need a mechanism for filtering non-conformance warnings that are exclusively due to the use of complex structures, thus narrowing the warnings to genuine deviations from the underlying template.

In Figure 3.13, we show how one can enhance TCC with such a mechanism using natural language parsing. First, all non-conformant requirements statements from the pipeline of Figure 3.8 are processed by a parser, with a parse tree such as the one in Figure 3.4(c) generated for each statement. Equipped with a parse tree, we can recognize complex noun phrases, in turn enabling us to distinguish between non-conformance due to the use of complex sentence structures from non-conformance due to genuine deviations from the template of interest.

```

1. Phase: DoMarkComplexNPs
2. Input: SyntaxTreeNode
3. Options: control = appelt
4.
5. Rule: MarkComplexNPs
6. (
7.   {SyntaxTreeNode.cat ==~ "NP"}
8. )
9. :label
10. -->
11. :label.NP= {}

```

**Figure 3.14.** JAPE script for marking complex NPs using a parse tree.

Figure 3.14 shows a JAPE script that can recognize complex noun phrases. The script searches for parse tree node annotations, denoted `SyntaxTreeNode`, that have NP as their category (line 7). Using the *appelt* control option (line 3) ensures that when multiple noun phrases are detected in the same text region, only the one that is the longest is marked.<sup>1</sup> For instance, running this JAPE script over the example of Figure 3.4(a) would mark the following two regions as NP: (1) “The information technology tools used in the design of systems performing safety functions”, (2) “safety implications on the end-product”.

Following the execution of the JAPE script in Figure 3.14, the requirements originally marked as non-conformant are reprocessed by the pipeline of Figure 3.13, but this time accounting for any new NP annotations generated for complex noun phrases. This second execution of the TCC pipeline filters out any non-conformance annotations caused by the limitation of text chunking in detecting complex phrases.

### 3.3.4 Checking NL Best Practices

In addition to checking template conformance, we use NLP for detecting and warning about several potentially problematic constructs, also called requirements smells [Femmer et al., 2014], that may be signs of vagueness or ambiguity in requirements statements. We build upon the requirements writing best practices by Berry et al [Berry et al., 2003]. Table 3.1 lists and exemplifies several constructs that we detect automatically. The automation is done through JAPE in a manner similar to how template conformance is checked.

## 3.4 Tool Support

We have implemented our approach in a tool named RETA (REquirements Template Analyzer). RETA has been developed as an application for the GATE workbench (<http://gate.ac.uk/>).

Figure 3.15 shows the overall architecture of RETA. Analysts may specify the requirements in the requirements authoring and management environment of their choice, e.g., Enterprise Architect (<http://www.sparxsystems.com.au>) or IBM DOORS ([www.ibm.com/software/products/ca/en/ratidoor/](http://www.ibm.com/software/products/ca/en/ratidoor/)). A glossary (if one exists) can optionally be provided to RETA to assist in the detection of noun phrases during the text chunking phase. The rules for checking conformance to

---

<sup>1</sup>JAPE’s different control options including *appelt* were discussed in Section 3.2.3.

**Table 3.1.** Potentially problematic constructs (from Berry et al [Berry et al., 2003]) detected through NLP.

Annotation	Potential Ambiguities	Example
Warn_AND	The “and” conjunction can imply several meanings, including temporal ordering of events, need for several conditions to be met, parallelism, etc.	The S&T module shall process the query data <b>and</b> send a confirmation to the database.  A temporal order is implied by the use of ‘and’.
Warn_OR	The “or” conjunction can imply “exclusive or”, or “inclusive or”.	The S&T module shall command the database to forward the configuration files <b>or</b> log the entries.  The inclusive or exclusive nature of ‘or’ is unclear.
Warn_Quantifier	Terms used for quantification such as all, any, every can lead to ambiguity if not used properly.	All lights in the room are connected to a switch. (example borrowed from Berry et al. [Berry et al., 2003])  Is there a single switch or multiple switches?
Warn_Pronoun	Pronouns can lead to referential ambiguity.	The trucks shall treat the roads before <b>they</b> freeze. (example borrowed from Berry et al. [Berry et al., 2003])  Does “they” refer to the trucks or the roads?
Warn_VagueTerms	There are several vague terms that are commonly used in requirements documents. Examples include userfriendly, support, acceptable, up to, periodically. These terms should be avoided in requirements.	The S&T module shall <b>support up to</b> five configurable status parameters.  “support” is a vague term. It is unclear whether “up to” means “up to and including”, or “up to and excluding”.
Warn_PassiveVoice	Passive voice blurs the actor of the requirement and must be avoided in requirements.	If the S&T module needs a local configuration file, it shall be created from the database system configuration data.  It is unclear whether the actor is S&T module, database, or another agent.
Warn_Complex_Sentence	Using multiple conjunctions in the same requirements sentence make the sentence hard to read and are likely to cause ambiguity.	The S&T module shall notify the administrator visually and audibly in case of alarms and events.  The statement may be interpreted as visual notification only for alarms and audible notification only for events.
Warn_Plural_Noun	Plural Nouns can potentially lead to ambiguous situations.	The <b>S&amp;T components</b> shall be designed to allow 24/7 operation without interruption.  Does this mean that every individual component shall be designed to be 24 / 7 or is this a requirement to be satisfied by the S&T as a whole?
Warn_Adverb_in_Verb_Phrase	Adverbial verb phrases are discouraged due to vagueness and the chances of important details remaining tacit in the adverb (e.g. frequencies, locations)	The S&T module shall <b>periodically poll</b> the database for EDTM CSI information.  The frequency of the periodic activity is unspecified.
Warn_Adj_followed_by_Conjunction	The adjective followed by two nouns separated by a conjunction, can lead to ambiguity due to the possible relation of adjective with just first noun or both nouns.	<b>compliant</b> hardware and software  Whether only hardware or both hardware and software have to be compliant is unclear.

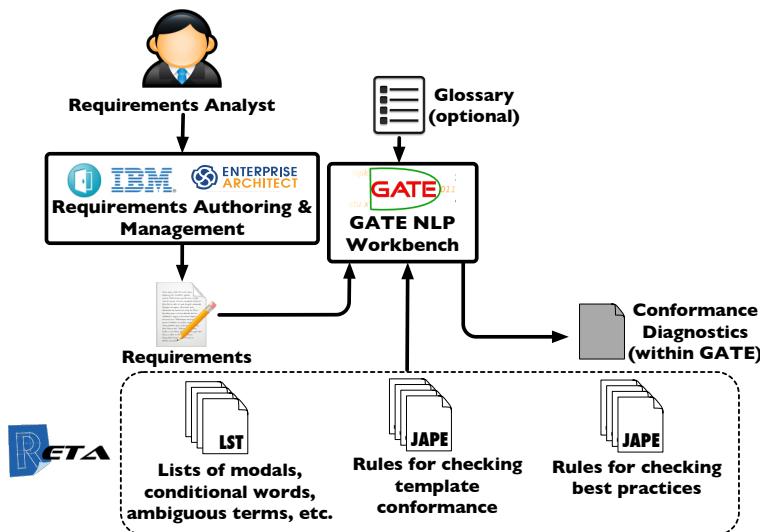


Figure 3.15. RETA tool architecture.

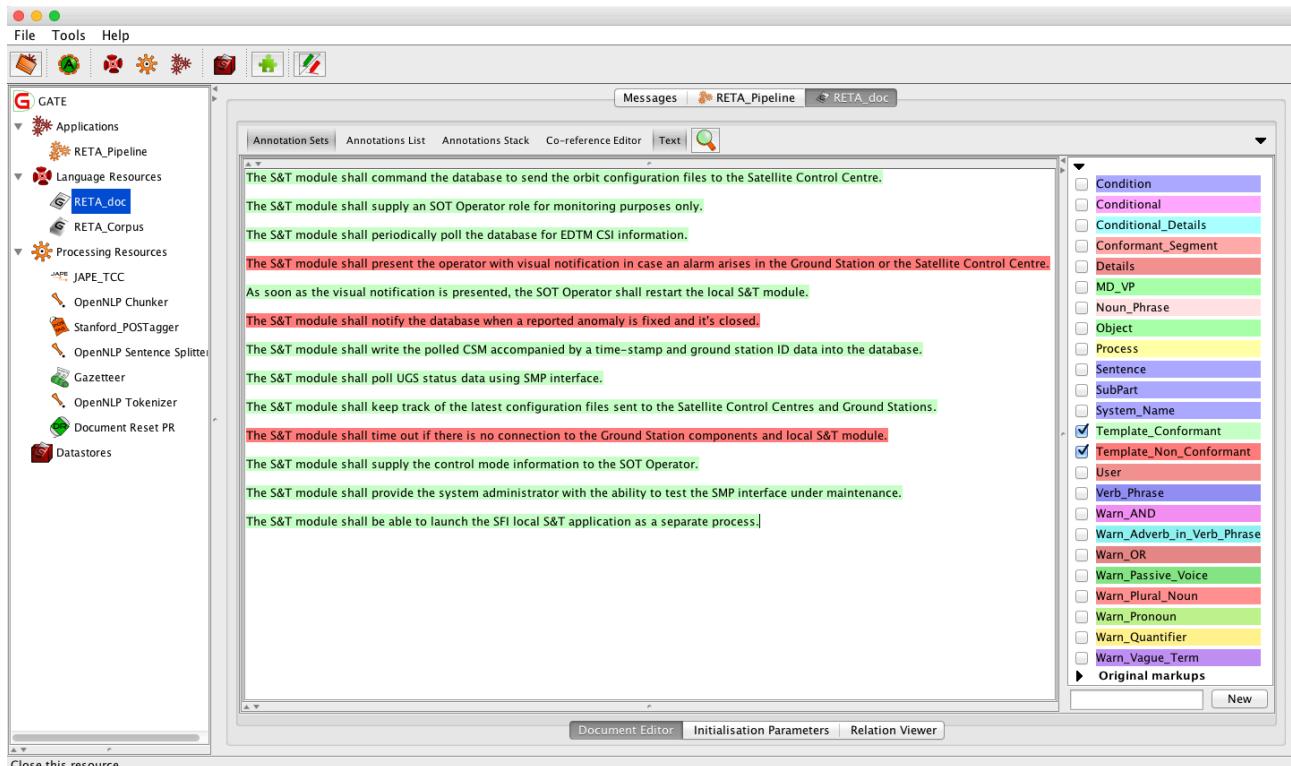


Figure 3.16. Snapshot of the annotations generated by RETA.

templates and best practices are provided as scripts written in GATE’s pattern matching language, JAPE. The various lists of terms used by these scripts, e.g., lists of template fixed terms, vague terms, conjunctions, modals and conditional words, are provided as customizable lists in GATE (gazetteers).

RETA uses GATE’s user interface for showing diagnostics about template non-conformance and deviations from requirements writing best practices. A snapshot of GATE’s interface after running the RETA application is shown in Figure 3.16. The requirements in the snapshot are drawn from one of our case studies (Case-A) discussed in Section 3.5. These requirements have been slightly altered from their original form to protect confidentiality.

In this snapshot, the left panel displays the various GATE resources (applications, language resources, and processing resources). The center panel displays the contents of the resource selected – here, the exemplar requirements document. The right panel displays the annotation labels for the current document. When an annotation is selected from the list, all its occurrences are highlighted over the document.

RETA consists of 30 JAPE scripts, containing in total approximately 800 lines of code. Out of these, 26 scripts are common between the implementation of Rupp’s and the EARS templates, suggesting that a large fraction of our implementation can be reused from one template to another. RETA further includes 10 lists (gazetteers) containing the keywords used by the scripts. Out of these, eight are common between the Rupp’s and the EARS template implementations.

To enable using GATE in realistic requirements development settings, we have implemented plugins to automatically export requirements written in IBM DOORS and Enterprise Architect. The IBM DOORS plugin has been implemented using a simple script written in DOORS Extension Language (DXL); the plugin for Enterprise Architect has been implemented in C# and is approximately 1000 lines of code. A demo version of RETA along with a screencast illustrating its use is available at:

<http://sites.google.com/site/retanlp/>

## 3.5 Evaluation

We have conducted four case studies for evaluating our approach. The case studies have been selected to address and balance several considerations. Most notably, these considerations include: coverage of different industrial domains, coverage of different templates, and ensuring realistic scale in terms of the number of requirements statements. In the remainder of this section, we discuss the design, execution, and results of our case studies.

### 3.5.1 Research Questions (RQs)

Our case studies aim to answer the following RQs:

**RQ1. What are optimal configurations for the NLP pipeline?** Text chunking uses several NLP modules, executed in a sequence over an input document. For each stage in the sequence, one needs to choose from a number of alternative implementations. The aim of RQ1 is to establish which

**Table 3.2.** The case studies used for evaluation.

Case	Description	Domain	Number of Requirements	Template Used
<b>Case-A</b>	Requirements for a software component in a satellite ground station	Satellites	380	Rupp's
<b>Case-B</b>	Requirements for a safety evidence management tool suite	Safety certification of embedded systems	110	Rupp's
<b>Case-C</b>	Requirements from Case-A restated in the EARS template.	Satellites	380	EARS
<b>Case-D</b>	Regulatory requirements for nuclear safety by the Finnish Radiation and Nuclear Safety Authority	Nuclear energy	890	EARS

combination of implementations produces the most accurate results. Precision, recall, and  $F$ -measure are used as metrics for assessing accuracy.

**RQ2. Does the absence of a glossary have an impact on the accuracy of our approach?** The glossary terms may be unknown or incomplete at the time one needs to check conformance to a template. RQ2 aims to investigate how the absence of a glossary affects the accuracy of our approach.

**RQ3. How effective is our approach at identifying non-conformance defects?** Our ultimate goal is to make it easier for practitioners to identify requirements that do not conform to a given template. With RQ1 establishing the level of accuracy to be expected from our approach, RQ3 aims to determine whether the accuracy is likely to be good enough from a practical standpoint.

**RQ4. Is our approach scalable?** In a realistic setting, one can be faced with hundreds and sometimes thousands of requirements. RQ4 aims to establish whether our approach runs within reasonable time.

In Section 3.5.6, we answer these RQs based on the results of our case studies, and further provide preliminary insights into the benefits of our approach from a practitioner's perspective.

### 3.5.2 Description of Case Studies

The case studies in our evaluation are the following:

- **Case-A** concerns a software component in a satellite ground station. The requirements for the component were written by professionals in the satellite industry using Rupp's template. This case study, a preliminary version of which was reported in [Arora et al., 2013a], was conducted in collaboration with SES TechCom – a satellite service provider. Case-A contains 380 requirements statements.
- **Case-B** concerns a tool suite for managing the safety information (safety evidence) used during the safety certification of embedded systems. The tool suite is currently under development in a European project named OPENCOSS (<http://www.opencoss-project.eu>). This case study was conducted in collaboration with requirements analysts from the OPENCOSS project. Similar to Case-A, Rupp's template was applied for writing the requirements. Case-B has 110 requirements statements.
- **Case-C** is a variant of Case-A, whereby all the 380 requirements in Case-A were transformed from Rupp's to the EARS template. Examples and details about the transformation are provided in Section 3.5.3.

- **Case-D** concerns safety requirements for nuclear facilities developed by the Finnish Radiation and Nuclear Safety Authority [STUK, 2016]. These requirements were written using the EARS template by requirements experts in collaboration with nuclear safety engineers. The experience from applying EARS to these requirements has been documented by the requirements authors [Uusitalo et al., 2011]. Case-D contains 890 requirements statements.

Table 3.2 summarizes key information about the case studies by providing for each case study a brief description, the domain in which the study was conducted, the number of requirements involved, and the requirements template used.

### 3.5.3 Data Collection Procedure

Data collection was performed in two phases: (1) documentation of requirements statements and identification of glossary terms; and (2) inspection of the requirements resulting from the first phase in order to determine which requirements are conformant and which ones are not. In Cases A, B, and D, Phase 1 was carried out by industry experts, except that in Case-D the glossary terms were not specified. In Case-C, the requirements were rephrased (from Case-A) by the researchers, with the glossary terms from Case-A reused as is. Phase 2 in all case studies was carried out by the researchers following the completion of Phase 1. We elaborate each of the two phases below:

**Phase 1.** The requirements in Case-A were written following two half-day training sessions, in which the researchers familiarized the participating industry experts with Rupp's template and how to use it. The researchers had no control over how the requirements were specified after these training sessions. In Case-B, the requirements originated from varied sources in the project's consortium. These requirements were documented using Rupp's template by expert engineers within the consortium, without involvement from the researchers.

Case-C requirements were derived from those in Case-A through a transformation. To this end, a systematic process was followed: (1) All non-conformant requirements (to Rupp's template) were carried over verbatim without being rephrased; (2) From the conformant requirements, those that did not have a conditional part were carried over verbatim as well and classified as *ubiquitous* requirements under EARS. (3) Conditional requirements were mapped to different requirements types in EARS based on the nature of the condition. For instance, requirements with temporal conditions were rephrased so as to conform to the *event-driven* requirements type in EARS. As an example, consider the following requirement in Rupp's template: “As soon as an unplanned outage is detected the S&T shall inform the SMP interface”. This requirement falls under the *event-driven* requirements type in EARS and is thus rephrased as follows in Case-C: “When an unplanned outage is detected the S&T shall inform the SMP interface”.

Note that the above transformation is *not* conformance-preserving. In particular, non-conformant requirements under Rupp's template may be deemed conformant under EARS. This happens due to the fact that in Rupp's template, it is mandatory to have an *⟨object⟩* slot following the *⟨process⟩* slot; whereas, in EARS no specific constraint exists regarding the presence or the position of an object in a requirements statement. For example, the requirement “The S&T shall present to the SOT operator the EDTM anomalies.” is not conformant to Rupp's template due to the object not being placed in the expected slot. This requirement is nevertheless conformant to EARS.

Finally, in Case-D, the requirements were written by experts with advanced training in the EARS template [Uusitalo et al., 2011]. The researchers were not involved in requirements specification.

In Cases A and B, the glossary terms were provided by the experts. Two considerations about glossary term identification were highlighted to the experts ahead of time: First, that for the purposes of our study, we did not require the experts to define the glossary terms, but only to identify them. Second, it was suggested to the experts that, when in doubt as to whether a particular term should be in the glossary, to include rather than exclude the term. These measures are important for RQ2, as the RQ aims to examine the effect of a glossary that is as complete as possible. Hence, we needed to mitigate the risk that the set of glossary terms would have major omissions due to the time pressure posed by having to define the terms. As noted earlier, for Case-C, we use the same glossary terms as Case-A. For Case-D, no glossary was provided as part of the requirements document and the researchers did not have access to the involved experts to elicit the glossary terms.

**Phase 2.** This phase is concerned with manually inspecting the requirements to determine which requirements (from Phase 1) are conformant to the underlying template. The data collected in this phase is used for calculating the effectiveness of automated conformance checking, as we discuss in Section 3.5.4. To increase the validity of the results, this phase was independently conducted by two different individuals. Subsequently, all discrepancies between the two inspectors were discussed and an agreement about conformance vs. non-conformance was reached in all cases. To ensure that the inspectors performed the inspection consistently with one another, an abstract inspection protocol, shown in Listing 1, was drawn up and followed by both inspectors.

An important factor during the inspection is whether the experts who wrote the requirements would like to admit only atomic noun phrases in the noun phrase slots (i.e.,  $\langle$ system name $\rangle$ ,  $\langle$ object $\rangle$ , and  $\langle$ whom? $\rangle$  for Rupp’s template and  $\langle$ system name $\rangle$  for the EARS template), or to further admit complex noun phrases, as discussed in Section 3.3.3. In Cases A, B, and C, the inspection would yield identical results irrespective of whether we admit complex noun phrases or not – an indication that the experts used only atomic noun phrases in the relevant slots. In contrast, in Case-D, the experts indicated that a conscious choice had been made to allow complex noun phrases in the  $\langle$ system name $\rangle$  slot. Accordingly, during our inspection of the requirements in Case-D, we accepted (grammatically-correct) complex noun phrases in the  $\langle$ system name $\rangle$  slot.

### 3.5.4 Analysis Procedure

Our analysis involves the execution of different configurations of NLP modules for text chunking and measuring how effective each configuration is in distinguishing requirements that conform to a template from those that do not.

#### 3.5.4.1 NLP Pipeline Configuration

To instantiate the pipeline of Figure 3.5, one needs to choose, for each step in the pipeline, a specific implementation from the set of existing alternative implementations. We narrow our investigation to the set of implementations in GATE. This decision is based on two considerations: First, GATE brings together a large collection of mature NLP technologies and provides a unified mechanism for integrating them through a generic annotation infrastructure. These characteristics of GATE make it

**Listing 1** Manual inspection protocol for template conformance.

---

```

1: Let  $R$  be the requirement being inspected for conformance to template  $T$  (either Rupp's or EARS).
2: Verify that  $R$  is a grammatically-correct sentence. Do not consider punctuation in determining correctness.
3: Verify that  $R$  uses an acceptable modal.
4: if  $R$  is conditional then
5:   Verify that the conditions appear only at the beginning of  $R$ .
6:   Verify that the conditions conform to the structure prescribed by  $T$ .
7: end if
8: if  $T$  is Rupp's template then
9:   Verify that  $\langle$ system name $\rangle$ ,  $\langle$ object $\rangle$ , and  $\langle$ whom? $\rangle$  (when applicable) are filled by noun phrases.
10:  Verify that  $\langle$ process $\rangle$  is filled by a verb phrase.
11: else if  $T$  is EARS then
12:   Verify that  $\langle$ system name $\rangle$  is filled by a noun phrase.
13:   Verify that  $\langle$ system response $\rangle$  starts with a verb phrase.
14: end if
15: if all criteria are fulfilled then
16:    $R$  is conformant to  $T$ ;
17: else
18:    $R$  is not conformant to  $T$ ;
19: end if

```

---

possible for us to experiment with several alternative solutions, beyond what would have been feasible in other existing NLP frameworks. Second, an important conclusion we would like to reach from our evaluation is how to build an accurate and at the same time practical tool. Focusing on a single NLP framework like GATE enables us to come up with concrete recommendations, without having to worry about the (likely) risk of interface incompatibilities between implementations that have not been already adapted to work together.

While our approach depends mainly on the annotations produced by the text chunking modules, i.e., Steps 5 and 6 in Figure 3.5, these two steps rely on the annotations produced in previous steps, i.e., Steps 1-4; therefore, the performance of Steps 5 and 6 ultimately relates to that of their previous steps. For this reason, we consider in our analysis not only different instantiations of Steps 5 and 6 but also different instantiations of Steps 1-4. Below, we list the different alternatives considered for each of the steps in Figure 3.5:

**Step 1 (2 alternatives):** ANNIE English Tokenizer [ANNIE, 2016], OpenNLP Tokenizer [OpenNLP, 2016]

**Step 2 (2 alternatives):** ANNIE Sentence Splitter [ANNIE, 2016], OpenNLP Sentence Splitter [OpenNLP, 2016]

**Step 3 (3 alternatives):** OpenNLP POS Tagger [OpenNLP, 2016], Stanford POS Tagger [StanPOS, 2016], ANNIE POS Tagger [Hepple, 2000]

**Step 4 (2 alternatives):** ANNIE Named Entity (NE) Transducer [ANNIE, 2016], OpenNLP Name Finder [OpenNLP, 2016]

**Step 5 (3 alternatives):** OpenNLP (NP) Chunker [OpenNLP, 2016], Multilingual Noun Phrase Extractor (MuNPEx) [MuNPEx, 2016], ANNIE Noun Phrase Chunker (an extension of Ramshaw & Marcus (RM) Noun Phrase Chunker [Ramshaw and Marcus, 1995])

**Step 6 (2 alternatives):** ANNIE Verb Group Chunker [ANNIE, 2016], OpenNLP (VP) Chunker [OpenNLP, 2016]

		Predicted (Automatic)	
		Conformant	Non-Conformant
Actual (Manual)	Conformant	True Negative (TN)	False Positive (FP)
	Non-Conformant	False Negative (FN)	True Positive (TP)

Figure 3.17. Confusion matrix for measuring accuracy.

Most of the above modules are based on machine learning. For all modules requiring training, we use the default training data (for English) that is distributed with GATE Release 8.0 [GATE User Guide, 2016].

The noun and verb phrase tags produced in Steps 5-6 are the basis for checking template conformance, as we described in Section 3.3. For Step 5, there is a choice to be made as to whether to include the glossary terms as an input. Therefore, we have a total of  $2 \times (2 \times 2 \times 3 \times 2 \times 3 \times 2) = 288$  different configurations to compare for cases A through C. For Case-D, since the glossary terms are unknown; we have only half the number of configurations, i.e., 144.

The annotations produced by each configuration is fed to the JAPE pipeline of Figure 3.8. For Case-D, since we would like to further admit complex noun phrases into the slots of the underlying template (as discussed in Section 3.5.3), the periphery pipeline of Figure 3.13 needs to be executed as well. We use the Stanford Parser [Klein and Manning, 2016] that is integrated into GATE 8.0 for the Parser step of the pipeline in Figure 3.13. The accuracy of TCC is subsequently analyzed using the accuracy measures discussed next.

### 3.5.4.2 Metrics for Measuring Accuracy

Our analysis of accuracy is based on *precision* and *recall*. These classification accuracy metrics are widely used in many areas, e.g., Information Retrieval (IR) [McGill and Salton, 1983], where one needs to measure the ability of an approach to correctly classify a set of objects into classes with certain properties. In our case, we are concerned with two classes: (1) template conformant and (2) template non-conformant. The simple confusion matrix shown in Figure 3.17 captures the possible errors that an automated conformance checker can make in the classification.

Precision is a metric for quality (low number of false positives) and is defined as  $TP/(TP + FP)$ . Recall is a metric for coverage (low number of false negatives) and is defined as  $TP/(TP + FN)$ . In most classification problems, including ours, an increase in precision comes at the cost of a decrease in recall and vice versa [Buckland and Gey, 1994]. To compare different NLP pipeline configurations while simultaneously accounting for both precision and recall, we use a metric, called *F*-measure [McGill and Salton, 1983], which computes the weighted harmonic mean of precision and recall. Depending on the context, one may want to place more emphasis on either precision or recall. In our study, recall is the primary factor as it is easier for analysts to rule out a small number of false positives than to go through a large document in search of false negatives. Hence, we use a definition of *F*-measure, known as  $F_2$ -measure, which gives more weight to recall than precision.  $F_2$ -measure is defined as:  $3 \times \text{Precision} \times \text{Recall} / (2 \times \text{Precision} + \text{Recall})$ .

**Table 3.3.** General statistics for the case studies.

Case	Template Conformant	Template Non-Conformant	Number of Glossary Terms	Requirements Types	Inter-rater Agreement (Cohen's Kappa)
Case-A	243 (64%)	137 (36%)	127	Autonomous	206
				User Interaction	35
				Interface	2
Case-B	98 (89%)	12 (11%)	51	Autonomous	44
				User Interaction	43
				Interface	11
Case-C	297 (78%)	83 (22%)	127 (reused from Case-A)	Ubiquitous	290
				Event-Driven	5
				Unwanted Behavior	2
Case-D	857 (96%)	33 (4 %)	0	State-Driven	0
				Optional Feature	0
				Complex	0
Case-D	857 (96%)	33 (4 %)	0	Ubiquitous	546
				Event-Driven	41
				Unwanted Behavior	72
Case-D	857 (96%)	33 (4 %)	0	State-Driven	22
				Optional Feature	150
				Complex	26

We note that other metrics and weights could have been used for combining precision and recall. Nevertheless,  $F_2$ -measure is standard when recall needs to be weighted higher than precision. We further note that we use classification accuracy metrics only for evaluation purposes. The end-users of our approach are not exposed to these metrics and do not need to make decisions based on them. The practical implications of these metrics for end-users are addressed in RQ3.

### 3.5.5 Results

This section describes the results of our case studies.

#### 3.5.5.1 Requirements Inspection and Glossary Elicitation

In Table 3.3, we provide various statistics about the case studies: the level of template conformance as established by the manual inspection protocol in Section 3.5.3, the number of glossary terms elicited from the experts, and the distribution of conformant requirements across the different requirements types in the underlying template. The table further shows the inter-rater agreement, expressed as Cohen's Kappa [Cohen, 1960], for the independent inspections conducted by two researchers. The conformance statistics in Table 3.3 are based on the mutually agreed-upon inspection results, after the differences between the two inspectors were discussed and resolved.

The requirements in Case-C, as mentioned previously, were derived from those in Case-A. The significant majority of (conformant) requirements in Case-C fall under the ubiquitous category, which is the simplest requirements type in EARS. This is because the requirements in Case-A were written without considering the EARS template. Hence, no conscious effort was made in Case-A to use the range of requirements types available in EARS. All cases, except Case-C, cover all the requirements types in their underlying template. Further, Case-C, as shown in Table 3.3, has a higher percentage of conformant requirements than Case-A (78% in Case-C versus 64% in Case-A). The reason for this increase in the rate of conformance is the absence of a mandatory  $\langle$ object $\rangle$  slot in the EARS template. Specifically, the requirements that are deemed non-conformant to Rupp's template in Case-A because of a missing or misplaced object are deemed conformant to EARS in Case-C.

Table 3.4. Results from the analysis of non-conformant requirements.

Non Conformance Type	Explanation	Example	Case	Percentage of Non Conformances
Minor Deviations	Requirement deviates only slightly from template's prescribed structure, e.g., by missing some of the fixed elements.	The S&T component shall provide the user with a function to view the network status.	Case-A	2 / 137 = 1.5%
			Case-B	0
			Case-C	0
			Case-D	0
Enumerations	Requirement concerns more than one object or functionality.	The state of the S&T module can be standby, active, or degraded.	Case-A	14 / 137 = 10.2%
			Case-B	2 / 12 = 16.7%
			Case-C	14 / 83 = 16.9%
			Case-D	5 / 33 = 15.2%
Missing or Misplaced Object	Exclusive to Rupp's template: The object slot is missing or placed at a non-prescribed position.	The OPENCOSS platform shall provide users with the ability to use in an assurance project evidence types that have been defined in another project.	Case-A	54 / 137 = 39.4%
			Case-B	6 / 12 = 50%
			Case-C	0
			Case-D	0
Incorrect Conditional Keyword	The conditional keyword is not among the ones prescribed by the template, e.g., WITHIN and AFTER.	After underwriting <sup>1</sup> is complete, if necessary the Insurance Officer shall perform underwriting <sup>2</sup> .	Case-A	2 / 137 = 1.5%
			Case-B	0
			Case-C	2 / 83 = 2.4%
			Case-D	0
Misplaced Conditions	Conditions appear in positions other than prescribed (which is the requirement's beginning).	The S&T module shall load a new configuration from the database as soon as the module receives a reloading request.	Case-A	61/137 = 44.5%
			Case-B	4 / 12 = 33.3%
			Case-C	61/ 83 = 73.5%
			Case-D	18 / 33 = 54.5%
Ill-formed Requirement	Requirement is grammatically incorrect or has some ill-formed slot.	The S&T shall periodically check the of the network elements by using the ping command.	Case-A	4 / 137 = 3%
			Case-B	0
			Case-C	4 / 83 = 4.8%
			Case-D	2 / 33 = 6.1%
Incorrect or missing modal	Requirement has an incorrect or missing modal in its verb phrase.	The approval of the final safety analysis report is a precondition for the endorsement of the application for an operating license.	Case-A	0
			Case-B	0
			Case-C	2 / 83 = 2.4%
			Case-D	8 / 33 = 24.2%

With regards to reasons for non-conformance, the majority of issues are explained by the factors listed in Table 3.4. As shown in the table, the most frequent reasons for non-conformance are misplaced or missing objects in Rupp's template, and misplaced conditions in both Rupp's and the EARS templates. These two factors collectively account for approximately 80% of non-conformance issues. The other factors are minor deviations, enumerations, incorrect conditional keywords, ill-formed sentences, and incorrect or missing modals.

For requirements containing enumerated objects, e.g., the example given for enumerations in Table 3.4, we did not advise the experts to take any remedial action to address non-conformance. This is because enumerated objects appear to be more naturally expressed using a sentence structure similar to our example, as opposed to the structure prescribed by the Rupp's and the EARS templates.

In the case of misplaced conditions, a closer examination was conducted to determine if one could naturally fit the affected requirements into the respective template, either by revising the sentence structures or by decomposing the affected requirements into finer-grained ones. Here, we observed what appears to be a limitation in the set of conditional keywords recommended by both Rupp's and the EARS template. Specifically, to be able to express a performance constraint, one often needs to use WITHIN, e.g., "WITHIN 2 seconds after a critical failure is detected, the S&T module shall

**Table 3.5.** Best pipelines in terms of  $F_2$ -measure.

Case	Tokenizer	Splitter	POS Tagger	Name Finder	Glossary?	NP Chunker	VP Chunker	Time (secs)	Precision	Recall	$F_2$ -measure
Case-A	--	--	Stanford	--	YES	MUNPEX	ANNIE	4 s (avg)	0.91	0.99	0.96
	OpenNLP	--	Stanford	--	NO	MUNPEX	ANNIE	3.5s (avg)	0.91	0.99	0.96
Case-B	--	--	OpenNLP	--	--	ANNIE	--	2s (avg)	1	1	1
	--	--	OpenNLP	--	--	OpenNLP	--	2.5s (avg)	1	1	1
Case-C	OpenNLP	--	OpenNLP	--	--	ANNIE	ANNIE	4.2s (avg)	0.94	0.98	0.97
Case-D	OpenNLP	ANNIE	OpenNLP	--	NO	OpenNLP	--	73.2s (avg)	0.94	1	0.98

Note: A “--” in the table means that the results are not sensitive to the choice of alternative used for that particular NLP module. See Section 5.4.1 for the set of alternatives for each module.

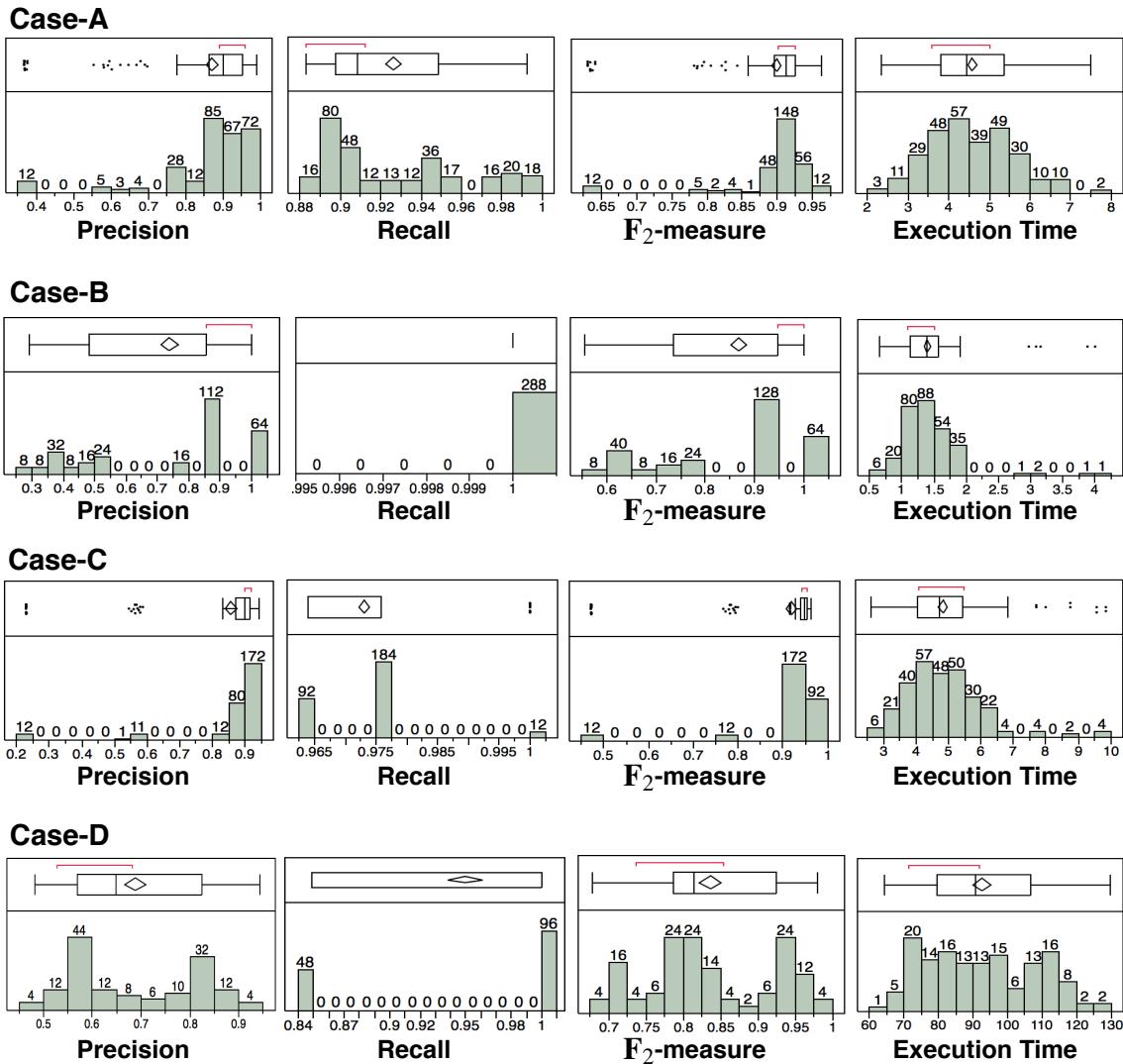
trigger the sound alarm on the main control panel”. During the inspection, a requirement like the above would be deemed non-conformant because the conditional keyword is not among the ones prescribed. We therefore propose that the conditional keywords in the two templates be extended with `WITHIN`. Our accuracy results, presented next in Section 3.5.5.2, are nevertheless calculated based on our original inspection results, i.e., as reported in Table 3.3, and irrespective of our own observation about conditional keywords.

### 3.5.5.2 Accuracy and Execution Time

The NLP pipelines discussed in Section 3.5.4.1 were executed for Cases A through D. For each pipeline, the classification accuracy metrics were computed as well as the time it took to execute the pipeline. In Table 3.5, we show the most accurate pipelines (best  $F_2$ -measure) for each case study. Descriptive statistics for precision, recall,  $F_2$ -measure, and execution time across *all* pipelines are given in the form of box plots in Figure 3.18. For Case-D, the accuracy results and execution times in Table 3.5 are inclusive of the periphery process for handling complex noun phrases. The accuracy gain from this periphery process is shown in Figure 3.19, where we contrast, through box plots, the accuracy with and without applying the periphery process. All experiments were conducted on a 2.3 GHz Intel Core i7 CPU with 8Gb of memory.

In Case-A and Case-C, there are 24 outliers in precision ( $< 0.7$ ), leading to 24 outliers in  $F_2$ -measure. All outliers have three features in common: the use of ANNIE Tokenizer, MuNPEX NP Chunker and the absence of a glossary. In 12 of the outliers, where precision is very low (in the 0.3–0.4 range), the poor outcome is due to MuNPEX NP Chunker being misled by incorrect tokenization produced by ANNIE Tokenizer. The result is that MuNPEX NP Chunker cannot correctly recognize the `<system name>` slot. In the remaining 12 outliers, the problem remains the same, except that OpenNLP NE Transducer corrects some of ANNIE Tokenizer’s mistakes, by recognizing the system name as an (atomic) named entity. Despite this, precision remains fairly low (in the 0.5–0.7 range for Case-A and 0.5–0.6 range in Case-C).

In Case-B and Case-D, there are no outliers, but there is significant variation in  $F_2$ -measure, brought about by the variation in precision. The variation in precision is largely explained by the

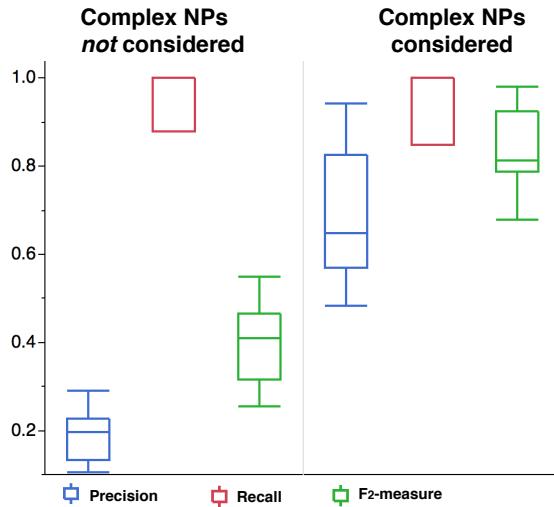


**Figure 3.18.** Box plots for classification accuracy metrics and execution times.

low number of non-conformant requirements. In other words, even a small number of false positives can have a considerable impact on precision and in turn on  $F_2$ -measure.

### 3.5.6 Discussion

**RQ1.** The best (i.e., most accurate) text chunking pipelines for Cases A through D were shown earlier in Table 3.5. Since the best choice differs across the case studies, one cannot recommend a single pipeline per se for use over a new (and unknown) requirements document. The results in Table 3.5 thus do not allow us to draw conclusions about the optimal NLP pipeline for text chunking. To come up with a general recommendation for the NLP pipeline, we need to pay attention to the impact that a particular NLP module can have on the outcome across all the pipelines it appears in. For example, a module that does not appear in the best pipeline but performs consistently well across all the pipelines it appears in may be preferred over a module that does appear in the best pipeline but also appears in some pipelines with poor results.



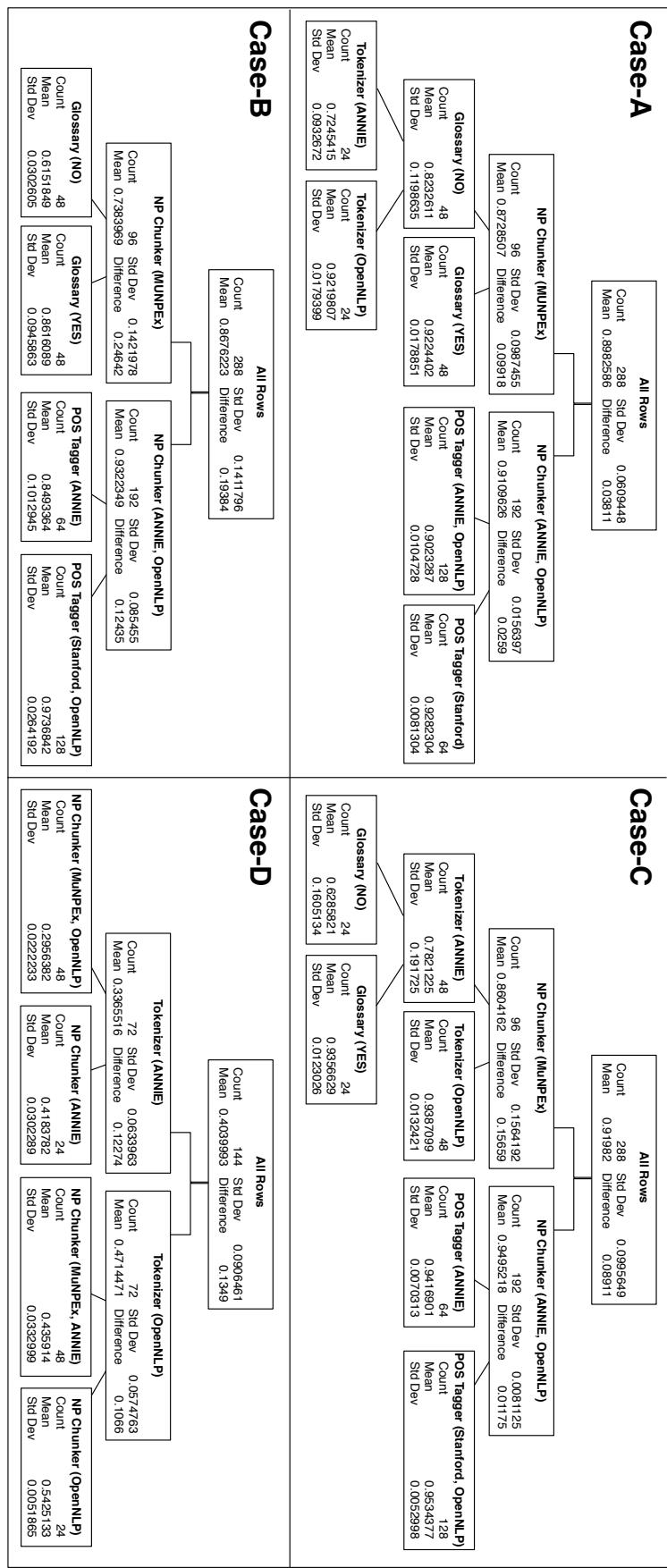
**Figure 3.19.** Accuracy results for Case-D with and without considering complex noun phrases.

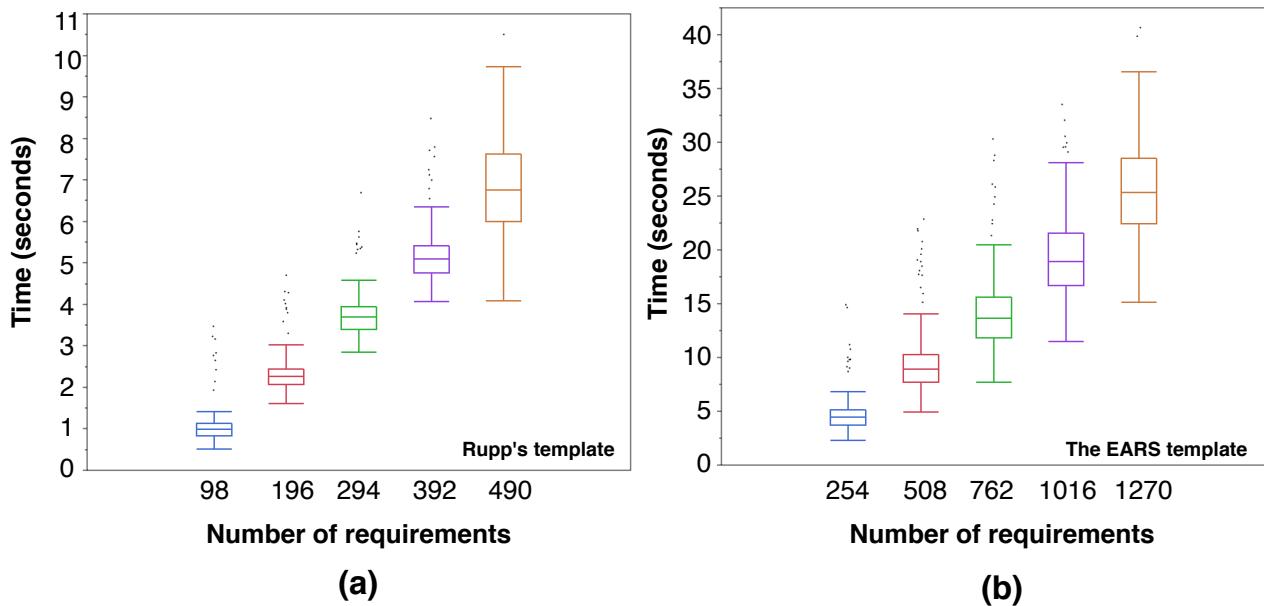
In more precise terms, we need to determine what modules cause the most variation in the accuracy metrics and avoid modules that cause a large degree of uncertainty. This analysis of variation is best conducted using a regression tree [Breiman et al., 1984]. A regression tree is built by partitioning a data set, e.g., NLP module combinations here, in a step-wise manner to obtain partitions that are as consistent as possible with respect to a certain criterion, e.g.,  $F_2$ -measure in our context.

In Figure 3.20, we show the regression trees for  $F_2$ -measure for our case studies. At each level in the tree, one NLP module is picked out and the pipelines are partitioned according to whether they use that module or not. The criterion for selection is to choose the module that would minimize the standard deviation across the branches that result after partitioning. In other words, the module that is most influential in explaining the variance in  $F_2$ -measure is selected. In each node of the tree, we show the count (number of pipelines), the mean and standard deviation for  $F_2$ -measure, and the difference between the smallest and largest  $F_2$ -measure observed in the partition. We note that the regression tree for Case-D has been constructed *without* applying the periphery process for handling complex noun phrases (see Figure 3.13). This is because the regression tree is meant for analyzing the sensitivity of accuracy to the text chunking NLP modules. The inclusion of the periphery process can undesirably alter the sensitivity results, because the process not only identifies complex noun phrases but also any atomic noun phrases that might have been missed by text chunking, thus masking text chunking errors.

As shown by Figure 3.20, in three out of the four cases, namely Cases A through C, the most critical decision concerns the choice of the NP Chunker module. In these three case studies, MuNPEEx NP Chunker performs well when the glossary terms are provided but does poorly on average otherwise. In Case-C (but not Case-A and Case-B), MuNPEEx NP chunker performs well in the absence of glossary terms, as long as OpenNLP Tokenizer is used for tokenization. Compared to MuNPEEx Chunker, ANNIE and OpenNLP NP Chunkers are less sensitive to the implementation choices for other NLP modules and thus introduce less variation. Within the ANNIE / OpenNLP Chunker branch in Cases A through C, the most critical decision concerns the POS Tagger, with Stanford and OpenNLP taggers achieving higher means.

The accuracy in Case-D is most sensitive to the choice of the Tokenizer, with OpenNLP leading to


 Figure 3.20. Regression trees for  $F_2$ -measure.



**Figure 3.21.** Execution time growth for checking conformance (a) to Rupp's template and (b) to the EARS template.

more accurate results than the ANNIE tokenizer. At the next level in regression tree, the most critical choice is that of the NP Chunker, with MuNPEx chunker yielding less accuracy than ANNIE and OpenNLP chunkers – consistent with Cases A through C.

The change in the most critical component being the Tokenizer in Case-D, as opposed to the NP Chunker in Cases A through C, is largely explained by the different tokenization behaviors of ANNIE and OpenNLP over certain patterns that were seen frequently in Case-D, most notably the use of cross-references in the requirements statements. In the case of the ANNIE tokenizer, the components of the numeric part of a cross-reference would be treated as separate tokens. For example, the ANNIE tokenizer would tokenize the cross-reference “Article 3.3.2.1” into eight tokens, one for the term “Article” and seven for the numeric part. OpenNLP would instead tokenize the cross-reference into two tokens, one for “Article” and one for the numeric part. Thus, in our context, misleading tokenization by the ANNIE tokenizer affects the behavior of the NP Chunker, making it the primary reason for wrongly-identified NPs, and eventually an incorrect delineation of the `<system name>` slot in the requirements statements. Subsequently, accuracy in Case-D is less affected by the choice of the NP Chunker than the Tokenizer.

Despite the above discrepancy between Case-D and Cases A through C, there is no inconsistency between the four case studies, in the sense that there are combinations of NLP modules that work well across all the case studies. Based on our analysis of the full regression trees for Cases A through D, we recommend the following modules for instantiating the text chunking pipeline:

- **Tokenizer:** OpenNLP Tokenizer
- **Sentence Splitter:** ANNIE Sentence Splitter OR OpenNLP Sentence Splitter
- **POS Tagger:** OpenNLP POS Tagger OR Stanford POS Tagger
- **NE Transducer:** ANNIE NE Transducer OR OpenNLP Name Finder
- **NP Chunker:** ANNIE (RM) NP Chunker OR OpenNLP NP Chunker

- **VP Chunker:** ANNIE VG Chunker OR OpenNLP VP Chunker

The above choices lead to the best overall results with little variation. Other alternatives may produce good results but may turn out to be too sensitive to the presence of certain other modules or conditions. For example, the MuNPEx NP Chunker should be used only in the presence of a (good) glossary.

**Table 3.6.** Average accuracy for recommended pipeline configurations.

Case	With Glossary			Without Glossary		
	Precision	Recall	F <sub>2</sub> -measure	Precision	Recall	F <sub>2</sub> -measure
<b>Case-A</b>	0.94	0.91	0.92	0.94	0.91	0.92
<b>Case-B</b>	0.93	1	0.98	0.93	1	0.98
<b>Case-C</b>	0.91	0.98	0.96	0.91	0.98	0.96
<b>Case-D</b>	-	-	-	0.85	0.96	0.92

If we narrow the text chunking pipeline configurations to those made up of the modules recommended above, we obtain 32 configurations that can be executed with or without a glossary. In Table 3.6, we show for each case study, the average accuracy of these 32 configurations. Comparing these accuracy levels against the best possible accuracy levels shown earlier in Table 3.5, we see an accuracy reduction of 4%, 2%, 1%, and 6%, respectively for Cases A through D.<sup>2</sup> We believe this reduction is small enough to be tolerated in exchange for the high stability of the results generated by our recommended configurations. In RQ3, we discuss the practical implications of these accuracy levels.

**RQ2.** As indicated by Table 3.6, as long as one uses the NLP modules recommended in RQ1, the presence of a glossary does not lead to accuracy gains. We therefore expect our approach to work with high accuracy even when the glossary terms are unknown.

**RQ3.** Table 3.7 shows for each of our case studies the expected number of false positives and false negatives, based on the number of requirements statements in the study and the average accuracy levels in Table 3.6 for our recommended pipelines.

The expected number of false positives is small across all case studies both in absolute numbers, as suggested by Table 3.7, and also as a percentage of the total number of non-conformances (Table 3.3). Specifically, this percentage is:  $8/137 = 5.8\%$  for Case-A,  $1/12 = 8.3\%$  for Case-B,  $8/83 = 9.6\%$  for Case-C, and  $6/33 = 18.2\%$  for Case-D. We thus anticipate that excluding false positives would comparatively take little effort.

The remaining question is how much of a quality problem false negatives pose. Ideally, one would like to avoid false negatives completely; however, this is practically infeasible as doing so will come at the expense of introducing a large number of false positives. Table 3.7 suggests that false negatives are few in absolute numbers. Further, when viewed as a percentage of the total number of requirements in each case study, false negatives constitute a small fraction:  $12/380 = 3.2\%$  for Case-A,  $0/110 = 0\%$

<sup>2</sup> As a technical remark, we note that the pipeline configurations in our study are deterministic, i.e., they produce the same results across different runs over the same input. In other words, we obtain a single accuracy value, as opposed to an accuracy distribution, for a given pipeline configuration over a given input. Due to the absence of random variation, we do not need statistical significance testing for comparing the accuracy results.

**Table 3.7.** Expected number of false positives and false negatives based on average accuracy levels (Table 3.6).

Case	Number of Requirements	False Positives	False Negatives
Case-A	380	8	12
Case-B	110	1	0
Case-C	380	8	2
Case-D	890	6	1

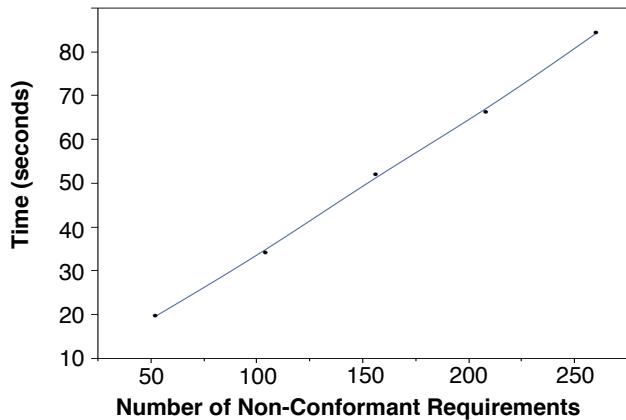
for Case-B,  $2/380 = 0.5\%$  for Case-C, and  $1/890 = 0.1\%$  for Case-D. For a large set of requirements in a real project, a manual inspection conducted under time pressure is unlikely to produce perfect results. Hence, such small fractions of false negatives are unlikely to outweigh the automation benefits of our approach, when compared to manual inspections.

**RQ4.** The main aspect of scalability that needs to be investigated in our context is whether our approach works within reasonable time over a large collection of requirements. Ideally, as the number of requirements grows, we expect the execution time to grow linearly as well. To analyze how execution times grow, we first combined the requirements sets written using the same template, i.e., we combined Case-A with Case-B requirements, and combined Case-C with Case-D requirements. The combination of Case-A and Case-B (Rupp's template) yields  $380 + 110 = 490$  requirements, and that of Case-C and Case-D (the EARS template) –  $380 + 890 = 1270$  requirements. We then randomized the order of the combined sets. Let *Case-AB* and *Case-CD* denote these sets after randomization.

From Case-AB, we built five requirements sets of increasing sizes: the first 98 (i.e.,  $490 / 5$ ) requirements of Case-AB, the first  $2 * 98$  requirements in Case-AB and so on. We built five similar sets for Case-CD: the first 254 (i.e.,  $1270 / 5$ ) requirements of Case-CD, the first  $2 * 254$  requirements of Case-CD, and so on until the last one with  $5 * 254 = 1270$  requirements. We then examined the execution time for TCC using different text chunking pipelines and against the growing number of requirements. In Figure 3.21, we show the execution time plots for each template. The results show a linear growth pattern in both plots, thus providing evidence that automatic TCC based on text chunking will scale linearly. Given such linear relation and the fact that processing the entire set of Case-AB and Case-CD requirements takes only a few seconds, we anticipate that our approach should be practical for much larger sets of requirements.

We further evaluated the scalability of the additional process for handling complex noun phrases in Case-D. To do so, we took the non-conformant requirements resulting from the execution of the least precise (i.e., worst) text chunking pipeline, followed by the JAPE pipeline of Figure 3.8. The rationale for using the least precise text chunking pipeline is to obtain the largest set of requirements that contain either complex noun phrases or potentially confusing segments for text chunking.

From the above process, we obtained a total of 260 non-conformant requirements statements. These statements were divided into 5 incremental sets, with cardinalities of  $260 / 5 = 52$ ,  $2 * 260 / 5 = 104$ , ..., and 260. We then subjected these sets to the process of Figure 3.13. In Figure 3.22, we show the execution time growth for the Parser step of the process, implemented via the Stanford Parser as discussed in Section 3.5.4.1. The “Mark Complex Noun Phrases” step of the process takes



**Figure 3.22.** Execution time growth for the Parser step envisaged by the process of Figure 3.13.

negligible time; the execution time growth for the second run of the pipeline of Figure 3.8, i.e., after the detection of complex noun phrases, is consistent with the execution times reported previously in Figure 3.21 and hence not shown. As suggested by Figure 3.22, while the Parser is computationally more expensive than text chunking, its execution time grows linearly as the number of requirements statements increases.

**Benefits from a practitioner's perspective.** Unless practitioners find our approach useful, they are unlikely to adopt it. It is therefore important to investigate practitioners' perceptions of the benefits of our approach. To do so, we draw on the qualitative reflections of a group of software and systems engineers at SES TechCom, with whom we have been collaborating on the research presented in this chapter. The reflections are based on the observations the researchers made throughout their interactions with the engineers and, in case of the expert in Case-A and Case-C, his hands-on experience applying Rupp's template and our tool.

The engineers' primary reason for interest in requirements templates and requirements writing best practices was to reduce ambiguity and vagueness as much as possible. They knew from experience that requirements containing vague terms or expressed using complex sentences were more likely to be the subject of clarification requests during formal requirements reviews. They believed that our tool would help reduce cost and overhead by enabling them to easily identify and address a sizable fraction of readability and vagueness issues before formal reviews took place. In a similar vein, the engineers found our tool to be useful as a training aid for requirements writing and, in the longer term, for establishing harmonized and industry-accepted requirements writing guidelines for their application domain.

Another important benefit the engineers noted with regards to templates is the flexibility that templates provide for requirements clustering. For example, conformance to Rupp's template would enable one to cluster the requirements based on the system name, the process verb being used, the object being processed, and so on. Such clustering facilitates the development of lower-stream artifacts, e.g., when one wants to orient the design or the test cases around requirements clusters. Since our approach automatically delineates different template slots in requirements, it can further automatically cluster the requirements based on the contents of different slots. In this respect, the engineers saw opportunities for cost and effort savings by applying our approach.

Given the limited scale to which our tool has been applied by our industry partner so far and the fact that we have not yet undertaken rigorous user studies, the benefits highlighted above are only suggestive but not conclusive. This being said, the positive experience reported by our industry partner and the fact that our tool could be successfully applied in real projects is promising and makes our approach worthy of future study.

## 3.6 Limitations and Threats to Validity

**Limitations.** Our approach tackles only structural conformance checking. This means that, as long as the syntax of a requirements statement follows what is prescribed by a given template, the statement will be deemed conformant. One cannot detect semantic mismatches using our approach, i. e. , the situation where the analyst’s choice of requirements type (and hence syntax) does not match the analyst’s intent. For example, an analyst may incorrectly frame an interface requirement as a user interaction requirement in Rupp’s template or confuse the event-driven and unwanted behavior types in the EARS template. Our approach is unable to detect such problems.

**Internal validity.** We tried to mitigate all foreseeable factors that could cause confounding effects. Particularly, learning effects from the tool were considered in the case study planning. The requirements experts had no exposure to the tool before finishing requirements specification and glossary terms elicitation. The use of case study data as test data for tool development was strictly avoided.

Another potential threat to internal validity is that the gold standard for evaluation was developed by the researchers. We took several mitigation actions to counter bias during the construction of the gold standard. Our tool was not used during the development of the gold standard to minimize influences on the reasoning of the researchers. A detailed protocol was drawn up and followed by the researchers for classifying requirements into conformant and non-conformant (see Section 3.5.3). Finally, the gold standard was constructed independently by two researchers, and the differences were then reconciled. Our inter-rater agreement analysis shows substantial or better agreement across our case studies (see Table 3.3), thus providing confidence about the quality of the gold standard that underlies our evaluation.

**Construct Validity.** The main consideration about construct validity has to do with what it means to conform to a template. The guidelines accompanying generic templates such as Rupp’s and EARS are intentionally abstract to ensure wide applicability. Subsequently, a certain degree of interpretation is required when one attempts to operationalize the process for conformance checking.

In our work, we opted for a *relaxed* definition of conformance, merely enforcing proper use of noun phrases and verb phrases. This is the most fundamental and yet the most complex aspect to handle for an automated tool. More restrictions can be considered for conformance, e. g. , ensuring absence of vague terms in the requirements. Our tool indeed already reports many such issues in the form of warnings (see Section 3.4). However, since such restrictions are easy to enforce automatically, i. e. , with a precision and recall of 100%, incorporating the restrictions into the definition of conformance provides “easy targets” for an automated tool to deem requirements as non-conformant. This can potentially conceal the mistakes that a tool might make in more complex operations, notably the detection of noun phrases and verb phrases. By having our definition of conformance focused on the

most basic criteria, we thus provide conservative estimates about the effectiveness of our approach. Therefore, even better results can be expected when the conformance is more constrained.

**External Validity.** We applied our approach to four case studies from different domains and using two different templates. We have further tried to remain as generic as possible in our treatment of template conformance. We believe that the consistency seen across the results of our case studies provides reasonable confidence about the generalizability of our approach. Further experimentation with requirements documents from other domains with even larger sizes would nevertheless be useful for improving external validity.

With regards to the benefits of our approach for practitioners (Section 3.5.6), the reflections presented in this chapter are those of a small number of domain experts working in a single application domain. Broader and more systematic user studies and surveys are necessary in the future for obtaining a more generalizable picture of the benefits and potential drawbacks of our approach.

## 3.7 Related Work

In this section, we compare with several strands of related work on requirements templates and applications of NLP to requirements.

### 3.7.1 Requirements Templates

Numerous requirements templates have been proposed over the years, e.g., by Rolland and Proix [Rolland and Proix, 1992], by Rupp [Pohl and Rupp, 2011], and by Mavin [Mavin et al., 2009]. These templates have been used both in academia and in industrial settings for specifying the requirements of complex systems [Daramola et al., 2012, Joseph et al., 2013, Slipper et al., 2013, Terzakis, 2013]. Our work in this chapter does not propose any new templates, but rather devises and empirically validates a scalable, flexible solution for automated checking of conformance to existing templates.

To examine the level of support for templates in existing tools, we conducted a tool review, guided by a recent requirements tool survey [de Gea et al., 2012] and a direct examination of the information sources that this survey builds upon. Specifically, we selected tools whose publicly-available feature descriptions include one or a combination of the following keywords: template, mold, template, syntax checking, linguistic analysis, and Natural Language Processing. We found nine tools that matched this criterion, namely, ARM [Wilson et al., 1997], Cradle [Cradle, 2016], DODT [Farfeleider et al., 2011], LEXIOR [LEXIOR, 2016], QuARS [Fabbrini et al., 2001], RQA [RQA, 2016], TIGER Pro [TigerPro, 2016], inteGREAT [inteGREAT, 2016], and visibleThread [Visible-Thread, 2016].

Upon closer examination of these tools, we identified two, DODT [Farfeleider et al., 2011] and RQA [RQA, 2016], offering automated support for checking template conformance. Our work differs from DODT in its focus: DODT concentrates on requirements transformation to achieve template conformance, whereas we focus on non-conformance detection. Further, in contrast to our work, DODT requires a high-quality domain ontology to be developed first, which as we argued earlier, is not a realistic expectation in most industrial development contexts.

As for RQA, while an investigation of the underlying conformance checking algorithm was not

possible due to the tool's proprietary nature, we observed that the tool's ability to identify sentence segments was impacted when the glossary terms were left unspecified. Our overall conclusion from the tool review is that although verifying template conformance is an important activity [Pohl and Rupp, 2011], limited automated assistance exists for it. In particular, tool support is lacking for settings where one has little control over the requirements authoring environments used by the analysts, e.g., when multiple organizations are involved in requirements writing. Having multiple organizations is also a factor that contributes to the difficulty of developing a glossary beforehand, thus making existing tools difficult to use. Our tool, in contrast, can be used in settings where little usable knowledge exists about how the requirements were written, and where all that is available for analysis are the requirements statements themselves.

### 3.7.2 NLP in Requirements Engineering

NLP has a long history of use in Requirements Engineering due to the prevalent use of natural language in the specification of requirements [Yilmaz and Yilmaz, 2011].

Quality assurance processes such as consistency checking represent one of the earliest areas where NLP has been applied to requirements. Gervasi and Nuseibeh [Gervasi and Nuseibeh, 2002] use domain-based parsing to enable checking of various formal properties over requirements. Gervasi and Zowghi [Gervasi and Zowghi, 2005] use part-of-speech tagging and parsing to automatically transform requirements into propositional logic and identify inconsistencies through logical reasoning. Kof et al. [Kof et al., 2010] develop an ontology-based technique for requirements validation using lemmatization and part-of-speech tagging.

NLP is further a cornerstone of automated requirements traceability, both for identifying inter-dependencies between requirements and for tracing requirements to lower-stream development artifacts such as design and source code. For example, Zou et al. [Zou et al., 2010] use phrase detection and similarity measures to enhance the requirements trace detection process. Duan and Huang [Duan and Cleland-Huang, 2007] combine similarity measures with clustering-based techniques to group candidate requirements trace results. Sundaram et al. [Sundaram et al., 2010] utilize similarity measures in a manner analogous to the above to improve requirements traceability detection. Sultanov and Hayes [Sultanov and Hayes, 2010] use tokenization and stemming for establishing traceability between requirements specifications at different levels of abstraction. Torkar et al. [Torkar et al., 2012] and Cleland-Huang et al. [Cleland-Huang et al., 2014] provide detailed reviews of the state-of-the-art on requirements traceability, including the range of NLP-based techniques used in this area.

Another area where NLP is commonly used is requirements ambiguity detection. Chantree et al. [Chantree et al., 2006] use morphological analysis for detecting noxious requirements ambiguities. Kiyavitskaya et al. [Kiyavitskaya et al., 2008b] use parsing for finding various potential syntactic and semantic ambiguities in requirements. Yang et al. [Yang et al., 2011] focus specifically on anaphoric ambiguities in requirements and develop a heuristics-based approach based on parsing for detecting this class of ambiguities. Femmer et al. [Femmer et al., 2014] use part-of-speech tagging, morphological analysis, and customized dictionaries for detecting patterns that are signs of potential quality defects in requirements.

Transforming NL requirements to models constitutes yet another important application of NLP in Requirements Engineering. Yue et al. [Yue et al., 2011] present a systematic literature review on the

approaches for transforming NL requirements into analysis models, further providing insights about how NLP is utilized in this context.

In addition to the general areas outlined above, where NLP is pervasive, there are various other Requirements Engineering tasks in which the use of NLP has been explored. For example, Zachos and Maiden [Zachos and Maiden, 2008] use text chunking for matching requirements to web-service descriptions. Kiyavitskaya et al. [Kiyavitskaya et al., 2008a] use part-of-speech tagging and parsing for generating various kinds of markup information over regulatory requirements. Holbrook et al. [Holbrook et al., 2009] utilize text chunking for assessing requirements satisfaction against different artifacts, such as design. Güldali et al. [Güldali et al., 2009] detect redundancies and implicit relationships between requirements using parsing and similarity measures. Falessi et al. [Falessi et al., 2013] apply and empirically compare different NLP-based strategies for identifying equivalent requirements. Guzman and Maalej [Guzman and Maalej, 2014] employ part-of-speech tagging, stemming and sentiment analysis for synthesizing user opinions about the requirements of mobile applications. Adedjouma et al. [Adedjouma et al., 2014] use tokenization and NLP pattern matching for detection and resolution of cross-references in legal texts. Arora et al. [Arora et al., 2014b] combine text chunking, similarity measures and clustering for extracting and grouping together requirements glossary terms.

None of the threads outlined above specifically address the problem that we tackle in this chapter, namely automatic checking of conformance to requirements templates. In addition, text chunking, which is the primary enabling technology for our approach has not yet been exploited widely in Requirements Engineering. Subsequently, limited empirical evidence exists about the effectiveness of text chunking over requirements documents. The empirical evaluation we report in this chapter provides insights into the effectiveness of various alternative implementations of text chunking, thus paving the way for the wider future application of text chunking in Requirements Engineering.

## 3.8 Conclusion

In this chapter, we presented an automated and tool-supported approach for checking conformance to requirements templates. The approach builds on a mature Natural Language Processing technique, known as text chunking. We reported on the application of the approach to four case studies from different domains, using two different templates – Rupp’s [Pohl and Rupp, 2011] and EARS [Mavin et al., 2009]. In this context, we evaluated and compared several text chunking solutions in terms of effectiveness and scalability for checking template conformance. Our evaluation results indicate that text chunking provides an accurate and scalable basis for template conformance checking. The study further shows that, within the range of alternatives considered, there exist several text chunking solutions with little sensitivity to the presence or absence of a requirements glossary. This makes it possible to automatically check and enforce the correct use of templates even in the (common) case where the glossary is partial or missing.

For future work, we plan to investigate whether it is feasible to automatically transform template grammars to NLP conformance checking pipelines. Automating this transformation enables one to directly derive a conformance checker from the design of a template, without the need to understand the NLP technology that underlies conformance checking.

# Chapter 4

## Glossary Terms Extraction and Clustering

Requirements glossaries provide an effective way to improve the accuracy and understandability of requirements statements, and to mitigate ambiguity [Lauesen, 2002, van Lamsweerde, 2009, Pohl, 2010]. A glossary makes explicit and provides definitions for the technical terms in a domain. A glossary may further provide information about the synonyms, related terms, and example usages of the domain terms. The lack of a requirements glossary can hinder teamwork and potentially jeopardize the success of a project [Young, 2004]. A key step in building a glossary is to decide upon the terms to include in the glossary and to find any related terms. Doing so manually is laborious, particularly for large requirements documents.

In this chapter, we develop an automated approach for extracting candidate glossary terms and their related terms from natural language requirements documents. Our approach differs from existing work on term extraction mainly in that it *clusters* the extracted terms by relevance, instead of providing a flat list of terms. We provide an automated, mathematically-based procedure for selecting the number of clusters. This procedure makes the underlying clustering algorithm transparent to users, thus alleviating the need for any user-specified parameters.

To evaluate our approach, we report on three industrial case studies, as part of which we also examine the perceptions of the involved subject matter experts about the usefulness of our approach. Our evaluation notably suggests that: (1) Over requirements documents, our approach is more accurate than major generic term extraction tools. Specifically, in our case studies, our approach leads to gains of 20% or more in terms of recall when compared to existing tools, while at the same time either improving precision or leaving it virtually unchanged. And, (2) the experts involved in our case studies find the clusters generated by our approach useful as an aid for glossary construction.

**Structure** The rest of this chapter is structured as follows: Section 4.1 emphasizes on the motivation for our work and presents our contributions. Section 4.2 provides background information and compares our work with related work. Section 4.3 presents our term extraction and clustering techniques. Section 4.4 describes our tool. Section 4.5 reports on the design and execution of our case studies. Section 4.6 discusses threats to validity. Section 4.7 concludes the chapter.

## 4.1 Motivation and Contributions

A glossary is an important part of any software requirements document. To ensure that requirements are written using a precise and consistent terminology, it is beneficial to build a glossary at the same time as when the requirements are being specified. This, however, is not always feasible due to time pressures. Too much upfront investment into the glossary may also be an issue from a cost-effectiveness standpoint, e.g., when the requirements are volatile and expected to change significantly as they are refined and prioritized.

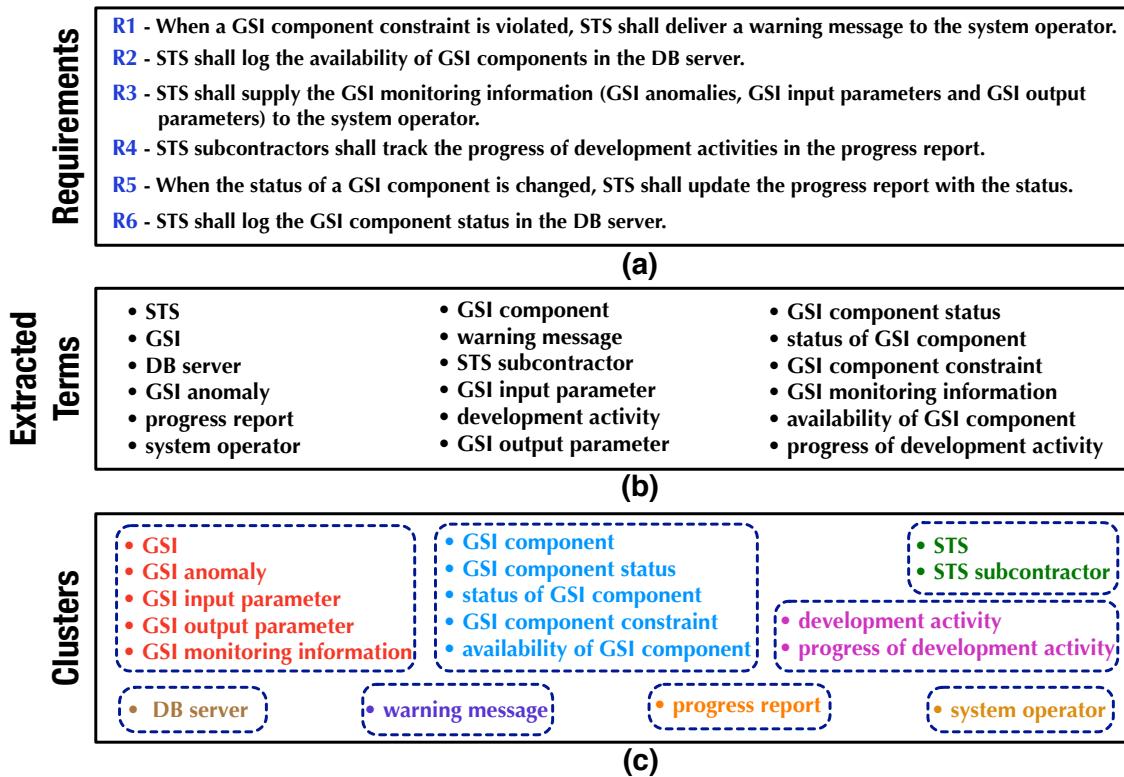
Consequently, requirements glossaries may be built *after the fact* and only when the requirements have sufficiently stabilized. This situation is, for example, reflected in the industrial requirements that are the subject of the case studies in this chapter (Section 4.5). To build a glossary after the fact, the terms to include in the glossary need to be extracted from the underlying documents. For large documents, a manual extraction of the terms may require a significant amount of effort, thus leaving less human resources for more complex tasks, e.g., writing the definitions for the glossary terms.

Our objective in this chapter is to automatically *extract* candidate glossary terms from natural language requirements and organize these terms into *clusters* of related terms. We illustrate the process using the example of Figure 4.1. In Figure 4.1(a), we show the requirements for which a glossary needs to be built. The requirements concern a satellite software component and represent a small fraction of a larger requirements document. The full document is the subject of one of the case studies in this chapter, as we discuss later. To protect confidentiality, the requirements have been sanitized without affecting their substance or structure. The abbreviations “GSI”, “STS”, and “DB” in the requirements stand for “Ground Station Interface”, “Surveillance and Tracking System”, and “Database”, respectively.

Given the requirements in Figure 4.1(a), we would like to first obtain a set of candidate terms such as those in Figure 4.1(b), and then group these terms into clusters such as those in Figure 4.1(c). By bringing together the related terms, these clusters can provide assistance to the analysts in a number of tasks, including deciding about the terms to include in the glossary, writing definitions for the glossary terms, and identifying potential consistency issues such as the use of variant phrases for referring to the same concept. An example of variant phrases in Figure 4.1 is “GSI component status” and “status of GSI component”.

Our work fits most closely with existing work on term extraction, which deals with automatic identification of the terminology in a given text corpus [Frantzi et al., 2000, Heylen and De Hertog, 2015]. Many strands of work exist on the subject, e.g., [Barker and Cornacchia, 2000, TOPIA, 2016, TextRank, 2016, TermRaider, 2016], to note a few. Despite term extraction being widely studied, existing tools are not tailored to requirements documents.

An important limitation that we have observed in generic term extraction tools is that these tools, when applied to requirements documents, yield poor recall, i.e., they miss a considerable number of glossary terms. This limitation is partly explained by filtering heuristics that are not suited to requirements documents. An example filtering heuristic is the exclusion, from the candidate glossary terms, of terms that are infrequent. While this heuristic is often necessary for extracting terms from large heterogeneous corpora, e.g., collections of books or articles, the heuristic is likely to filter important terms that, despite having a low frequency of appearance, would warrant a precise definition



**Figure 4.1.** (a) Example requirements from a satellite component, (b) candidate glossary terms extracted from the requirements, (c) candidate terms organized into clusters.

when used in a requirements document. Poor recall is in another part due to the absence of heuristics for combining adjacent phrases under certain conditions. For example, consider the variants “GSI component status” and “status of GSI component”, mentioned earlier. One would expect that these variants will be treated the same way by a term extractor. However, the tools we have investigated (Section 4.2.4) would detect only the former because it is a single noun phrase, but not the latter, because it is a combination of two noun phrases.

We take steps in this chapter to address the above limitation. More importantly, what contrasts our work from the existing work on term extraction is that, instead of producing a flat list of candidate terms, our approach produces clusters of related terms. As we argue more precisely later in the chapter based on our empirical results, we believe that clusters provide a more suitable basis than flat lists for performing the tasks related to glossary construction.

We propose an automated solution for extracting and clustering candidate glossary terms in requirements documents. Specifically, we make the following contributions:

(1) We develop a term extraction technique using a well-known natural language processing (NLP) task called *text chunking* [Ramshaw and Marcus, 1995]. In particular, we are interested in noun phrase (NP) chunks in requirements documents. NPs correspond closely to candidate glossary terms. We propose complementary heuristics to address limitations in a naive application of chunking.

(2) We develop a technique to cluster candidate glossary terms based on syntactic and semantic simi-

larity measures for natural language. An important consideration with regard to clustering is selecting an appropriate number of clusters. To avoid the need for users to specify this number in an arbitrary manner, we provide automated guidelines for estimating the optimal number of clusters.

**(3)** We report on the design and execution of three industrial case studies, as part of which we also analyze the perceptions of the industry experts involved in the case studies about the usefulness of our approach. Our evaluation results notably suggest that: (1) Our term extraction technique outperforms, by a factor of 20% or more, all the generic term extraction tools compared with in terms of recall, while at the same time also outperforming all but one of these tools in terms of precision. Our term extraction technique, when compared to the best generic alternative, has lower precision in two of our case studies. Nevertheless, the precision loss is negligible (-0.9% in one case study and -1.2% in the other) and significantly outweighed by gains in recall. And, (2) the experts find the clusters computed by our clustering technique useful as an aid for defining the glossary terms, for identifying the related terms, and for detecting potential terminological inconsistencies.

**(4)** We develop a tool, named REGICE, implementing our term extraction and clustering techniques. The tool is available at <https://sites.google.com/site/svvregice/>. To facilitate the replication of our empirical results using REGICE or other tools, we provide on REGICE's website the material for one of our case studies, whose requirements document is public-domain.

## 4.2 Background

This section presents background on the key technologies used in the chapter. The section further reviews and compares with related strands of work.

### 4.2.1 Text Chunking

Our approach builds on text chunking. In the previous chapter (Section 3.2.2), we already explained the text chunking pipeline (Figure 3.5). This pipeline may be realized in many different ways as there are alternative implementations available for each of the modules in the pipeline. In the previous chapter, we further evaluated 144 possible combinations of module implementations for building the text chunking pipeline.

In this current chapter, we employ one of the most accurate text chunking pipeline instantiations identified in the previous chapter. The choice of module implementations used is given in Section 4.4. Briefly, we recall that text chunking is aimed at recognizing the grammatical segments (phrases) of sentences, including among others, Noun Phrases (NPs), Prepositional Phrases (PPs), and Verb Phrases (VPs). For the purpose of this chapter, we are interested only in NPs. An *NP* is a segment that can be the subject or object in a sentence. According to Justeson and Katz [Justeson and Katz, 1995], NPs account for 99% of the terms in technical glossaries. The remaining 1% are typically VPs. Our term extraction technique does not return any VPs in its results, since the benefits of doing so are significantly outweighed by the overhead of having to manually filter undesirable VPs from the results.

## 4.2.2 Computing Similarities between Terms

To cluster candidate terms, we need a degree of relatedness between term pairs. We define this degree using syntactic and semantic similarity measures, as we describe next.

### 4.2.2.1 Syntactic Similarity Measures

Syntactic (similarity) measures calculate a score for a given pair of terms based on the terms' string content. These measures are usually normalized to a value between 0 and 1, with 0 indicating no similarity at all, and 1 indicating perfect similarity, i.e., string equivalence. For example, syntactic measures would normally yield a high score for the terms "GSI component" and "GSI component status" because of the large textual overlap between the terms. In our empirical evaluation (Section 4.5), we consider 12 syntactic measures: *Block distance* [Gomaa and Fahmy, 2013], *Cosine* [Gomaa and Fahmy, 2013], *Dice's coefficient* [Gomaa and Fahmy, 2013], *Euclidean distance* [Gomaa and Fahmy, 2013], *Jaccard* [Cohen et al., 2003], *Char-Jaccard* [Cohen et al., 2003], *Jaro* [Cohen et al., 2003], *Jaro-Winkler* [Cohen et al., 2003], *Level Two (L2) Jaro-Winkler* [Cohen et al., 2003], *Levenstein* [Manning et al., 2008], *Monge-Elkan* [Monge and Elkan, 1997], and *SoftTFIDF* [Cohen et al., 2003]. These measures are described in Table 4.1.

Syntactic measures can be classified into three categories: *distance-based*, *token-based*, and *corpus-based* [Cohen et al., 2003]. Distance-based measures calculate a score for a given pair of terms by finding the best sequence of edit operations to convert one term into the other. *Levenstein* is an example of such measures. Token-based measures work by treating each term as a bag of tokens and then matching the tokens of different terms. An example such measure is *Cosine*. Corpus-based measures enhance either distance-based or token-based measures by accounting for the characteristics of the corpus from which the terms are drawn. For example, to calculate a similarity score for a pair of terms, *SoftTFIDF* considers the frequency of the terms' constituent tokens in the corpus where the terms appear. The standard implementation of *SoftTFIDF* uses *Jaro-Winkler* (a distance-based measure) as a similarity predicate over tokens.

### 4.2.2.2 Semantic Similarity Measures

Given a pair of words, semantic (similarity) measures calculate a similarity score based on the meaning of the words in a dictionary. Similar to syntactic measures, semantic measures are often normalized. For example, most semantic measures would produce a non-zero score for the words "communication" and "transmission" because of the semantic affinity between these two words. In this chapter, we experiment with eight semantic measures: *HSO* [Hirst and St-Onge, 1998], *JCN* [Jiang and Conrath, 1997], *LCH* [Leacock and Chodorow, 1998], *LIN* [Lin, 1998], *LESK* [Banerjee and Pedersen, 2003], *PATH* [Pedersen et al., 2004], *RES* [Resnik, 1995], and *WUP* [Wu and Palmer, 1994]. These measures are described in Table 4.2.

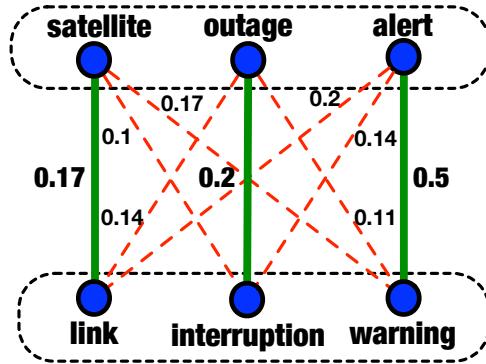
To generalize semantic measures from single-word terms (i.e., tokens) to multi-word terms, one must define a strategy for combining token-level similarity scores. To this end, we adopt the strategy used by Nejati et al. [Nejati et al., 2012]: Given a pair of (multi-word) terms, we treat the terms as bags of tokens. We then calculate similarity scores for all token pairs using the semantic measure of choice. In the next step, we compute an optimal matching of the terms' constituent tokens. A matching is optimal if it maximizes the sum of token-level similarity scores. Finally, we calculate the

**Table 4.1.** Description of the syntactic similarity measures considered in our empirical evaluation.

Measure	Description
Block Distance	Computes similarity between terms by considering them as vectors and calculating the traversal distance between the vectors in a two-dimensional plane represented by the vectors.
Cosine	Computes similarity between terms by transforming the terms into vectors and then calculating the angle between the vectors.
Dice's coefficient	Computes similarity between terms by finding the tokens that are in common and then dividing the number of common tokens by the total number of tokens in the terms.
Euclidean	Computes similarity between terms by transforming the terms into vectors and calculating the normalized difference between them.
Jaccard	Computes similarity between terms by finding the common tokens and dividing the number of common tokens by the number of tokens in the union of bags of words between the terms.
CharJaccard	A variation of Jaccard similarity that works at the level of characters rather than tokens.
Jaro	Computes similarity between terms using the number of common characters in each token of the terms.
Jaro-Winkler	An extension of Jaro similarity which combines the Jaro score with the length of the common prefix between terms.
Level-Two (L2) Jaro-Winkler	A hybrid measure that computes a normalized score for all the possible substrings (at the token level) of two terms using a secondary measure (Jaro-Winkler).
Levenshtein	Computes similarity between terms based on the minimum number of character edits (insertions, deletions, and substitutions) required to transform one term into the other.
Monge-Elkan	Computes similarity between terms by matching all the individual tokens of the terms and normalizing the similarity score based on the similarity of tokens.
SoftTFIDF	Computes similarity between terms based on a secondary measure, combined with the frequency of the single-word constituents of the terms in a corpus. We use Jaro-Winkler as the secondary measure. In our context, the corpus is the set of <i>all</i> candidate terms. The intuition is that two terms are more similar if they share several single-word constituents with comparable frequencies.

**Table 4.2.** Description of the semantic similarity measures considered in our empirical evaluation.

Measure	Description
HSO	Computes similarity between words by finding a short path in the is-a (vertical) and has-part (horizontal) relation chains (as specified in WordNet) that does not change direction too often. An example of an is-a relation is arm “is-a” limb. An example of a has-part relation is arm “has-part” forearm.
RES	Computes similarity between words based on the information content of the least common subsumer (LCS) of the words in an is-a hierarchy. Information content is the degree of specificity of words. For example, “car” is a more specific word than “vehicle”. Therefore, “car” has a higher information content value than “vehicle”. The LCS is the most specific concept that two words share as an ancestor in an is-a hierarchy. For example, the LCS of “car” and “boat” is “vehicle”.
JCN	Computes similarity between words by augmenting RES (above) with the individual information content of the words.
LIN	Computes similarity between words in the same manner as JCN (above) but with a slightly modified similarity formula.
LESK	Computes similarity between words by quantifying the overlap between the different dictionary meanings of the words.
PATH	Computes similarity between words based on the shortest path between them in an is-a hierarchy.
LCH	Computes similarity between words based on the shortest path between all the meanings of the words.
WUP	Computes similarity between words based on the depth of the words and their LCS in an is-a hierarchy.



**Figure 4.2.** Example of semantic similarity calculation for multi-word terms.

normalized sum for the optimal matching and take the result as the similarity score for the given terms. More precisely, given a pair  $(t_1, t_2)$  of terms, the (term-level) semantic similarity score,  $\mathcal{S}(t_1, t_2)$ , is:

$$\mathcal{S}(t_1, t_2) = 2 \times \frac{\text{sum of token similarity scores in optimal match}}{N_1 + N_2}$$

where  $N_1$  and  $N_2$  denote the number of tokens in  $t_1$  and  $t_2$ .

Figure 4.2 illustrates the calculation of a similarity score for the terms “satellite outage alert” and “link interruption warning”. Here, the token-level similarity scores, shown on the lines that connect the tokens, were calculated using the *PATH* measure. The optimal matching between the tokens is shown using solid lines. Based on this optimal matching, the similarity score for the terms in question is:  $2 \times (0.17 + 0.2 + 0.5) / (3 + 3) = 0.29$ .

### 4.2.3 Clustering

Clustering refers to the task of grouping related objects in a manner that the objects in the same cluster are more similar to one another than to the objects in other clusters [Aggarwal and Reddy, 2013].

To devise an accurate technique for clustering glossary terms, we need to address two important factors. First, we need to select a suitable clustering algorithm from the alternatives available. Second, we need to define a strategy for tuning the input parameters of the selected algorithm. Having such a strategy is essential in order to avoid the end-user from having to make ad-hoc decisions about the input parameters. In particular, virtually all clustering algorithms require the number of clusters to be given *a priori* as an input parameter. Naturally, the value of this parameter varies from one requirements document to another, depending on the document’s size and complexity. If a poor choice is made about the number of clusters, the accuracy of clustering may be severely compromised.

Below, we review the clustering algorithms examined in this chapter as well as the criterion that we use for estimating the optimal number of clusters for a given requirements document.

#### 4.2.3.1 Clustering Algorithms

We experiment with three well-known clustering algorithms, *K-means*, *Agglomerative Hierarchical* and *EM*, to determine which one(s) are the most accurate in our application context. Our choice of these algorithms is motivated by their prevalent use for clustering of natural-language content [Aggarwal and Reddy, 2013]. Below, we briefly outline these algorithms. Further details can be found in clustering and data mining textbooks, e.g., see [Aggarwal and Reddy, 2013].

**K-means** partitions a given set of data points (in our context, candidate terms) into  $K$  clusters, where  $K$  is an a-priori-given number. Briefly, *K-means* attempts to assign each data point to a cluster in a way that maximizes the similarity between the individual data points in each cluster and the center of that cluster, called a *centroid*. The centroids and the cluster membership functions are iteratively improved until convergence, i.e., when a fixpoint is reached.

**(Agglomerative) Hierarchical Clustering** groups a set of data points by building a tree-shaped structure, called a *dendrogram*. The data points constitute the dendrogram's leaf nodes. A dendrogram is not one set of clusters, but rather a cluster hierarchy. Each non-leaf node in a dendrogram represents a cluster made up of all leaf nodes (data points) that are descendants of the non-leaf node in question. The algorithm starts by assigning each data point to its own cluster. It then finds the closest pair of clusters, i.e., the pair with the largest similarity, or dually the shortest distance, and merges the cluster pair into one cluster. This process is repeated until all the data points have been absorbed into a single cluster, represented by the root node of the dendrogram. There are several alternatives ways for computing the similarity, or dually, the distance, between two clusters during hierarchical clustering. In this chapter, we consider eight alternatives: *average link*, *centroid link*, *complete link*, *McQuitty's link*, *median link*, *single link*, *Ward.D link*, and *Ward.D2 link*. A description of these alternatives is provided in Table 4.3.

**Table 4.3.** Description of the alternative criteria considered in our empirical evaluation for computing cluster distances when hierarchical clustering is applied.

Measure	Description
Single link	Computes the distance between two clusters as the least distance between any constituent terms.
Complete link	Computes the distance between two clusters as the maximum distance between any constituent terms.
Average link	Computes the distance between two clusters as the average distance between all pairs of the constituent terms of two clusters.
Centroid link	Computes the distance between two clusters as the distance between their centroids.
Median link	Computes the distance between two clusters as the distance between their medians.
Ward.D link	Computes the distance between two clusters as the sum of squared deviations from terms to centroids.
Ward.D2 link	A variant of ward.D (above).
McQuitty's link	Computes the distance between a new cluster, resulting from the merge of two existing ones, and other clusters by averaging the distances from both parts of the new cluster. The merge will take effect when the two parts as a pair have the least average distance to other clusters.

Given a dendrogram, one can obtain a single set of clusters either by cutting the dendrogram at a given height,  $H$ , or by splitting the dendrogram into a given number,  $K$ , of clusters. Either way, the value of the respective parameter has to be specified by the user. We were unable to identify generalizable guidelines to help decide the value of  $H$  in the first option above. In this chapter, we therefore consider only the second option, i.e., splitting into a prespecified number of clusters.

**Expectation Maximization (EM)** is a statistical clustering algorithm. In this chapter, we use a common variation of EM, where it is assumed that a set of observations –in our case, the similarity degrees between candidate terms– is a combination of a given number,  $K$ , of multivariate normal distributions [Fraley and Raftery, 2012]. Each distribution, characterized by its mean and covariance matrix, represents one cluster. The EM algorithm attempts to approximate the  $K$  individual distributions, so that their combination best fits the observations. Here,  $K$  corresponds to the number of clusters. Initially, the EM algorithm chooses random values for the means and covariance matrices of the distributions. The algorithm then iterates through the following two steps until convergence:

- *Expectation step*: Given the means and covariance matrices of the  $K$  distributions, estimate the membership probability of each data point (candidate term) in each distribution.
- *Maximization step*: Estimate new values for the means and covariance matrices of the  $K$  distributions, using maximum-likelihood estimation [Scholz, 1985].

Once the algorithm converges, each data point is assigned to the cluster in which it has the largest membership probability.

#### 4.2.3.2 Choosing the Number of Clusters

As we stated earlier, choosing an appropriate number of clusters, denoted  $K$  in Section 4.2.3.1, is imperative for the accuracy of clustering. If  $K$  is too large, closely related terms will be scattered over different clusters rather than being grouped together; if  $K$  is too small, we will be left with clusters in which the terms have little or no relationship to one another.

Several metrics exist for estimating the optimal number of clusters. Among these, Bayesian Information Criterion (BIC) [Schwarz et al., 1978] is one of the most reliable [Divakaran, 2009, Aggarwal and Reddy, 2013]. BIC is computed as a byproduct of EM clustering. Nevertheless, the metric is also commonly used for estimating  $K$  in both  $K$ -means and hierarchical clustering [Divakaran, 2009, Aggarwal and Reddy, 2013]. We use BIC as the basis for assigning a value to  $K$  in our approach.

Briefly, BIC is a measure for comparing statistical models with different parameterization methods, different numbers of components, or both [Fraley and Raftery, 2012]. When developing a statistical model to fit given data, one can improve the fit by adding additional parameters. This may however result in overfitting. To avoid overfitting, BIC penalizes model complexity so that it may be maximized for simpler parameterization methods and smaller numbers of components (clusters, in our case) [Fraley and Raftery, 2012]. The larger the BIC value is, the better the fit and consequently the better the selected number of clusters. Adapting BIC to our application context is the subject of one of our research questions; see RQ3 in Section 4.5.6.3.

#### 4.2.4 Related Work

In this section, we discuss and compare with several other strands of related work in the areas of term extraction, clustering, and NLP.

#### 4.2.4.1 Term Extraction

We organize our review of term extraction into two parts: (1) general literature and tools, and (2) relevant research in the subject field of this chapter, i.e., Requirements Engineering.

**General literature and tools.** Term extraction has been studied in many domains and under numerous titles, including terminology identification, terminology mining, term recognition, term acquisition, keyword extraction, and keyphrase detection [Heylen and De Hertog, 2015]. Term extraction approaches can be broadly classified into three categories [Pazienza et al., 2005]: *linguistic*, *statistical*, and *hybrid*.

Linguistic approaches aim at specifying patterns for detecting terms based on their linguistic properties, e.g., their POS tags. For example, Bourigault [Bourigault, 1992] describes a linguistic approach for extracting terms by eliminating certain grammatical patterns like pronouns and determiners, and then using regular expressions based on POS tags to extract certain combinations of NPs. Aubin and Hamon [Aubin and Hamon, 2006] use a combination of chunking and parsing to extract both simple and complex NPs.

Statistical approaches select terms based on statistical measures such as frequency and length. For example, Jones et al. [Jones et al., 1990] develop a statistical approach for identifying keywords by assigning ranks to word sequences in a document in such a way that frequently-occurring sequences which have many frequently-occurring words receive a high rank. In a similar vein, Matsuo and Ishizuka [Matsuo and Ishizuka, 2004] use the co-occurrence frequency of words and of sequences of words for identifying keywords.

Hybrid approaches are combinations of linguistic and statistical ones. For example, Barker et al. [Barker and Cornacchia, 2000] first employ text chunking for identifying the NPs in a given text, and subsequently filter out terms that are unlikely to be keywords based on frequency and length.

Our approach is a linguistic one. We do not use statistical measures because these measures primarily serve as filters. Speaking in terms of classification accuracy metrics, filtering improves precision, i.e., it decreases false positives. However, improvements in precision may come at the cost of losses in recall, i.e., increases in false negatives. For large and heterogeneous corpora, e.g., book and article collections, online commentary and – in the case of software – repositories of development artifacts, statistical measures, notably frequencies, provide a useful indicator for the importance of terms. Over such corpora, filtering based on statistical measures is often essential in order to achieve reasonable precision. In requirements documents, however, every individual statement is expected to have a clear and non-redundant purpose. Therefore, terms in requirements documents, regardless of their statistical characteristics such as frequencies, have the potential to bear important content. Consequently, using statistical filters over requirements documents is likely to have a significant negative impact on recall. In our work, we take a conservative approach towards filtering. In particular, we do not filter any terms on statistical grounds. The only filter we apply is a linguistic one over common nouns, as we explain in Section 4.3.

With regard to tool support for term extraction, several generic tools are already available, including the following:

- *JATE (Java Automatic Term Extraction toolkit)* [Zhang et al., 2008] implements several term extraction techniques developed and used by the Information Retrieval (IR) community.
- *TextRank* [TextRank, 2016] is a general text processing tool, with term extraction being one of its constituent parts. Extraction is performed based on POS tags, and an undirected graph in which edges represent pairwise relationships between terms based on their level of co-occurrence.
- *TOPIA* [TOPIA, 2016] is a widely-used Python library for term extraction based on POS tags and simple statistical measures, e.g., frequencies.
- *TermRaider* [TermRaider, 2016] is a term extraction module implemented as a plugin for the GATE NLP Workbench [GATE, 2016]. TermRaider uses advanced heuristics based on POS tags, lemmatization, and statistical measures.
- *TermoStat* [Drouin, 2003] is a term extraction tool based on POS tags, regular expressions, and frequency-based measures.

All the aforementioned tools are based on hybrid techniques. As we demonstrate in Section 4.5, over requirements documents, our proposed term extraction technique yields better recall than these tools without compromising precision. Our work is further distinguished from these tools in that it clusters the extracted terms based on relatedness.

**Term extraction in Requirements Engineering.** Term extraction has been tackled previously in Requirements Engineering. Aguilera and Berry [Aguilera and Berry, 1990] and Goldin and Berry [Goldin and Berry, 1997] present frequency-based methods for identifying terms that appear repeatedly in requirements. They refer to these terms as “abstractions” which are likely to convey important domain concepts. Popescu et al. [Popescu et al., 2008] extract terms from restricted natural language requirements using parsing and parse relations. Zou et al. [Zou et al., 2010] use a POS tagger for extracting single- and double-word noun phrases, and then filter the results based on frequency measures and certain heuristics. Kof et al. [Kof et al., 2010] use POS tags, named-entity recognition, parsing, and heuristics based on sentence structures for extracting domain-specific requirements terms. Dwarakanath et al. [Dwarakanath et al., 2013] use parsing for extracting the phrases of requirements documents and then filter the results based on heuristics and frequency-based statistics. And, Ménard and Ratté [Ménard and Ratté, 2015] extract domain-specific concepts from business documents (including requirements) using POS tag patterns and various heuristics.

These earlier threads have helped us in better tailoring our term extraction technique to requirements. The main technical novelties contrasting our work from the above are the following: (1) We use text chunking for identifying candidate terms; text chunking is more accurate and generalizable than pattern-based techniques based on POS tags, and more robust and scalable than parsing for phrase detection [Song et al., 2006]. And, (2) we apply clustering for grouping candidate terms. Furthermore, and for reasons discussed earlier, our term extraction technique does not make use of statistical filters.

#### 4.2.4.2 Clustering

Grouping together (clustering) related terms has been studied in the field of Information Retrieval. Existing approaches rely on pre-defined patterns of POS tags for identifying relatedness. Daille [Daille, 2005] uses the prefix POS tags of NPs for identifying adjectival and prepositional modifications.

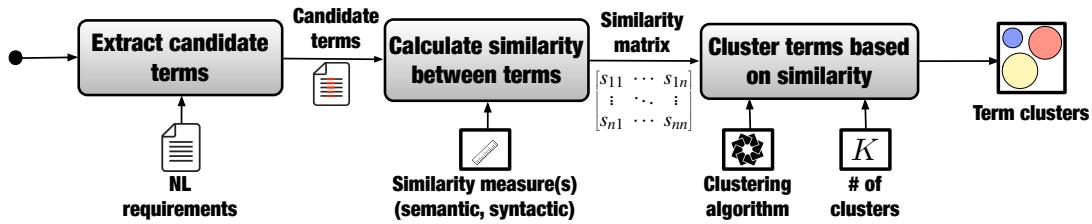


Figure 4.3. Approach overview.

For example, their approach would group the terms “package” and its adjectively-modified form “biodegradable package”. Similarly, Bourigault and Jacquemin [Bourigault and Jacquemin, 1999] group related terms based on patterns of noun modifiers. For example, their approach would group the terms “cylindrical cell” and “cylindrical bronchial cell”. The main difference between our approach and the above is that, instead of patterns, we use syntactic and semantic similarity measures for detecting relevance. For unrestricted natural-language content, an exhaustive enumeration of all patterns of interest is very difficult; pattern-based approaches are therefore prone to incompleteness. Our approach does not suffer from this issue. Furthermore, our approach can systematically deal with morphological and semantic relatedness, which existing pattern-based approaches do not address sufficiently.

In the field of Requirements Engineering, clustering has been already applied for a variety of purposes. Ferrari et al. [Ferrari et al., 2013] cluster requirements statements in order to organize them into cohesive sections within requirements documents. Arafeen and Do [Arafeen and Do, 2013] use requirements clustering as an enabler for test case prioritization. Chen et al. [Chen et al., 2005] cluster related requirements for building product-line feature models. Duan et al. [Duan and Cleland-Huang, 2007] apply and empirically evaluate the usefulness of different clustering techniques for grouping related development artifacts (requirements, test cases, classes, etc.) and supporting traceability. They further provide guidelines for selecting the number of clusters in this application context. Finally, Mahmoud [Mahmoud, 2015] uses clustering for classifying non-functional requirements and tracing them to functional requirements.

Our application of clustering is guided by the same principles as in the above threads of work. Nevertheless, these threads do not use clustering to achieve the same end goal as ours, which is grouping together candidate glossary terms. A critical prerequisite for applying clustering effectively is to identify, for a specific analytical task, a suitable combination of a clustering algorithm and similarity measures. Doing so requires empirical studies that focus on the task at hand. To our knowledge, an empirical study similar to the one in this chapter does not exist for the task of building glossaries.

#### 4.2.4.3 Natural Language Processing

In addition to term extraction, reviewed in Section 4.2.4.1, there are several other Requirements Engineering tasks in which NLP has been used for automation. These tasks include, among others, identification of inconsistencies and ambiguities [Gervasi and Nuseibeh, 2002, Gervasi and Zowghi, 2005, Chantree et al., 2006, Kiyavitskaya et al., 2008b, Yang et al., 2011, Femmer et al., 2014, Misra, 2015], requirements tracing [Duan and Cleland-Huang, 2007, Sultanov and Hayes, 2010, Sundaram et al., 2010, Torkar et al., 2012, Cleland-Huang et al., 2014, Pruski et al., 2015], requirements change analysis [Arora et al., 2015b], detection of redundancies and implicit requirements relations [Güldali

et al., 2009, Falessi et al., 2013], extraction of models from requirements [Yue et al., 2011, Vidya Sagar and Abirami, 2014], identification of use cases [Holbrook et al., 2009], markup generation for legal requirements [Adedjouma et al., 2014, Zeni et al., 2015], enforcement of requirements templates [Arora et al., 2015a], synthesis of user opinions about features [Guzman and Maalej, 2014, Maalej and Nabil, 2015], and requirements identification [Riaz et al., 2014].

The NLP techniques we use in this chapter are not new to the Requirements Engineering community. Nevertheless, the NLP techniques underlying our approach, notably text chunking and semantic similarity measures, have not been systematically studied alongside clustering before. Furthermore, empirical studies that investigate the effectiveness of NLP over industrial requirements remain scarce. The case studies we report on in this chapter take a step towards addressing this gap.

## 4.3 Approach

Figure 4.3 shows an overview of our approach. Given a (natural-language) requirements document, we first construct a list of candidate glossary terms. In the next step, we compute a similarity matrix for the extracted terms. In the third and final step, we cluster the terms based on their similarity. The rest of this section elaborates each of these steps.

### 4.3.1 Extracting Candidate Glossary Terms

Using text chunking, this step extracts a set of candidate glossary terms from a given requirements document. As explained in Section 4.2.1, from the results of text chunking, we need only the NPs. Following text chunking, all the extracted NPs are processed and cleared of determiners, pre-determiners, cardinal numbers, and possessive pronouns. For example, “the system operator” is reduced to “system operator”. Furthermore, plural terms are transformed into singular terms using lemmatization. For example, “GSI anomalies” is transformed into “GSI anomaly”.

Subsequently, we refine the list of terms by applying the heuristics listed in Table 4.4. The first heuristic in the table aims at re-establishing the context that may have been lost for some NPs. Specifically, text chunking decouples concepts from their attributes / subparts connected by “of” or a possessive ‘s. For example, the phrase “status of GSI component” gives rise to two NPs: “status” and “GSI component”. However, “status” is unlikely to be useful as a term outside its context. To capture this intuition, we add to the list phrases of the form: *NP of NP* and *NP’s NP*.

The second heuristic adds to the list of terms: (1) any all-capital token appearing within some NP, and (2) any continuous sequence of tokens marked as proper nouns (NNPs) by the POS tagger within the boundary of an individual NP. For example, the token “GSI” in “GSI component” will be added to the list as an independent term and so will “Ground Station Interface” if an NP such as “Ground Station Interface component” is already on the list. This is despite the fact that “GSI” and “Ground Station Interface” may never appear in the document as NPs. This heuristic is targeted at ensuring that potential abbreviations and named entities will have dedicated entries in the list of terms.

The last heuristic in Table 4.4 is for filtering common nouns. By a common noun, we mean a single-word noun, e.g. “status”, that is found in an (English) dictionary. We use the WordNet dictionary [Fellbaum, 1999] for word lookup operations. The rationale for filtering common nouns

**Table 4.4.** Heuristics applied to the results of text chunking.

Heuristic	Description	Example
<i>NP of NP / NP's NP</i>	The combination of two NPs separated by “of” or a possessive ‘s is added to the list of terms.	“status of GSI component” <i>The second NP, i.e., “GSI Component” provides the context for the first NP, i.e., “status”.</i>
<i>Special tokens and sequences of proper nouns</i>	Abbreviations and sequences of proper nouns (marked as NNP by the POS tagger) within individual NPs are added as independent entries to the list of terms.	“GSI component” / “Ground Station Interface component” <i>The abbreviation “GSI” and the sequence of proper nouns “Ground Station Interface” are extracted and added as independent entries to the list of terms.</i>
<i>Common nouns</i>	Single-word phrases that have a meaning in an English dictionary are filtered out.	“status” <i>This NP is unlikely to contribute to the glossary unless coming alongside its context, e.g., as in “status of GSI component”, or is capitalized to signify a probable proper noun.</i>

is that these nouns are often generic and polysemous, and thus, outside their context, unlikely to contribute to the glossary [Bourigault and Jacquemin, 1999]. Single-word nouns that are not found in a dictionary or are capitalized will be retained in the list of terms.

Finally, we remove any duplicates from the list of terms. Figure 4.1(b) shows the list of terms derived from the requirements of Figure 4.1(a) through the process described above.

Measuring the accuracy of our term extraction technique and comparing the accuracy to that of generic term extraction tools is the subject of one of our research questions; see RQ1 in Section 4.5.6.1.

### 4.3.2 Computing Similarities between Terms

This step computes a similarity matrix to capture the degree of relatedness between every pair of candidate terms extracted in the previous step. To compute this matrix, we consider three alternative strategies [Arora et al., 2015b]:

1. *syntactic only*, where a similarity,  $\mathcal{S}_{syn}(t, t')$ , is computed for every pair  $(t, t')$  of terms using a syntactic measure, e.g., SoftTFIDF;
2. *semantic only*, where a similarity,  $\mathcal{S}_{sem}(t, t')$ , is computed for every pair  $(t, t')$  of terms using a semantic measure, e.g., JCN;
3. *combined syntactic and semantic*, where, given a syntactic measure  $syn$  and a semantic measure  $sem$ , we take, for every pair  $(t, t')$  of terms,  $\max(\mathcal{S}_{syn}(t, t'), \mathcal{S}_{sem}(t, t'))$ . Using  $\max$  is motivated by the complementary nature of syntactic and semantic measures [Nejati et al., 2012, Achananuparp et al., 2008].

Choosing the best strategy from the above and the specific similarity measures to use are addressed by RQ2 and RQ4; see Sections 4.5.6.2 and 4.5.6.4.

### 4.3.3 Clustering Terms

In this step, we cluster the candidate terms based on their degree of relatedness. The inputs to this step are the similarity matrix (or dually, the distance matrix in the case of hierarchical clustering),

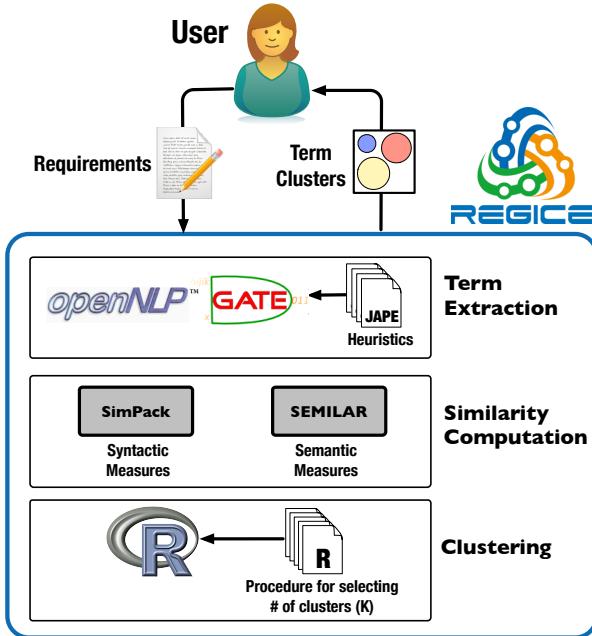


Figure 4.4. Tool Overview.

the choice of clustering algorithm to use, and the number of clusters,  $K$ , to generate. As the result of clustering, the terms are grouped into  $K$  partitions. For example, Figure 4.1 (c) shows a partitioning of the terms in Figure 4.1 (b) with  $K = 8$ . The clustering algorithm applied here is EM and the similarity measure used is SoftTFIDF alone (i.e., without an accompanying semantic measure).

In our empirical evaluation of Section 4.5, we investigate all the key questions related to tuning clustering for use in our application context. Specifically, identifying the most accurate clustering algorithm(s) is addressed in RQ2 and RQ4. Choosing a suitable  $K$  is tackled in RQ3; see Section 4.5.6.3.

## 4.4 Tool Support

We implement our approach into a tool named REGICE (REquirements Glossary term Identification and ClustEring tool). The components of REGICE are shown in Figure 4.4.

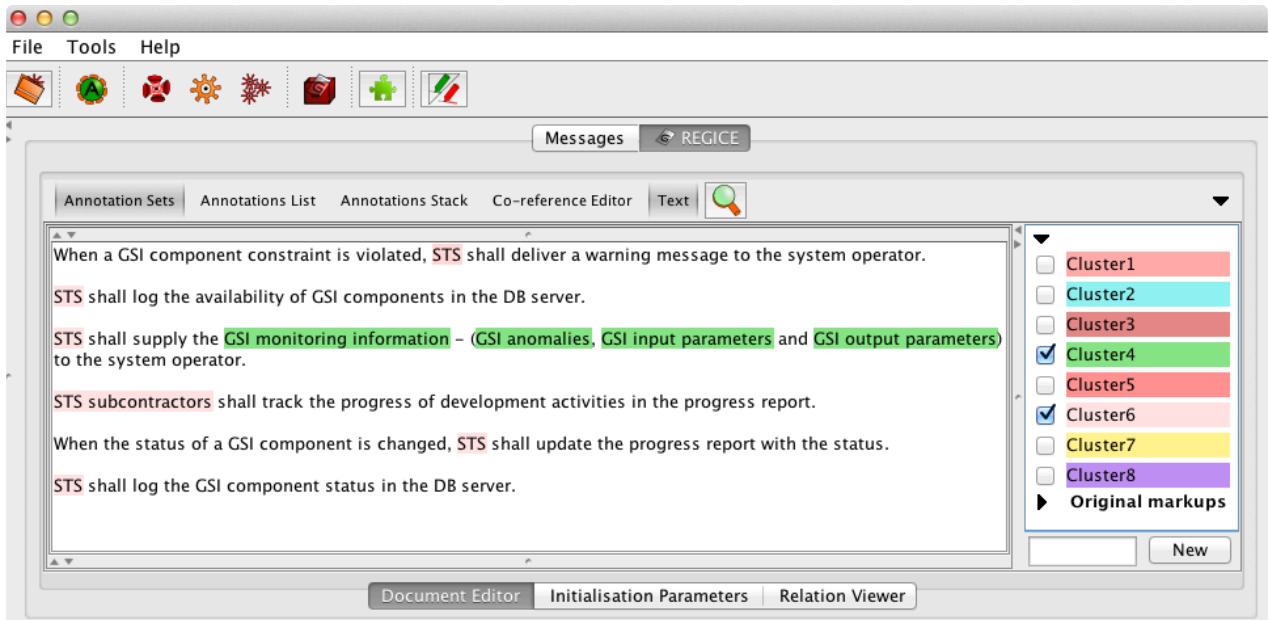
First, the requirements provided by the user are processed by a text chunker in the GATE NLP Workbench [GATE, 2016]. GATE is an infrastructure built over a large collection of heterogeneous NLP technologies, making it possible for these technologies to interact and work together. Within GATE, there are several alternatives for implementing the text chunking pipeline discussed in Section 4.2.1. Among the alternatives, we use OpenNLP [OpenNLP, 2016]. This choice is based on a comparative study in our previous work [Arora et al., 2015a], where we found the OpenNLP chunking pipeline to be one of the most accurate and robust alternatives over requirements documents. The heuristics we apply for refining the results of text chunking (Section 4.3.1) are implemented using scripts written in GATE’s regular expression language, JAPE (Java Annotation Patterns Engine).

We use SimPack [SimPack, 2016] for computing syntactic similarities and SEMILAR [Rus et al., 2013] for computing semantic similarities between the extracted terms. Both libraries are Java-based.

The default syntactic measure in REGICE is SoftTFIDF. No semantic measure is used by default, although the user has the option to choose any of the semantic measures provided by SEMILAR. Our default choices are based on our empirical observations (from RQ4) in Section 4.5.

For clustering and computing the BIC (Section 4.2.3.2), we use the R statistical toolkit [R, 2016]. The default clustering algorithm in REGICE is EM, again based on our empirical observations (from RQ4). The R library used for clustering depends on the choice of the clustering algorithm. *K*-means and all variants of hierarchical clustering are done using the `stats` package. EM clustering and BIC computation are done using the `mclust` package [Fraley and Raftery, 2012].

The number of clusters to generate is determined automatically using R scripts that implement the guidelines derived from our empirical results (RQ3). The user has the option to override the automatic recommendation for the number of clusters and provide a different number of clusters.



**Figure 4.5.** Screenshot of REGICE (implemented in GATE [GATE, 2016]) with two computed clusters highlighted.

REGICE provides two alternative ways for presenting the computed clusters: (1) Writing the clusters as labeled sets in a file, similar to what is shown in Figure 4.1 (c), or (2) visually highlighting the clusters over the requirements through GATE’s user interface. We illustrate this user interface in the screenshot of Figure 4.5. The requirements in this screenshot are from the example of Figure 4.1. As shown by the right panel of the screenshot, each cluster is represented as an annotation type. When a cluster (annotation type) is selected, all the terms in that cluster are highlighted in the document. This visual representation has the advantage that it preserves the context where each term in a given cluster appears.

The components of REGICE have been integrated together via glue code written in Java. REGICE, including the R and JAPE scripts, is approximately 2000 lines of code excluding comments and third-party libraries. The tool is available at: <https://sites.google.com/site/svvregice/>.

Case	Description	Domain	Number of Requirements
<b>Case-A</b>	Requirements for a software component in a satellite ground station	Satellites	380
<b>Case-B*</b>	Requirements for a safety evidence management system	Safety certification of embedded systems	110
<b>Case-C</b>	Requirements from a data dissemination network solution for satellites.	Satellites	138

\* The material for Case-B is available on our tool's website (see Section 4).

**Figure 4.6.** Description of case studies.

## 4.5 Evaluation

We evaluate our term extraction and clustering techniques over three industrial case studies. In this section, we elaborate the research questions that motivate our evaluation and report on the design, execution and results of the case studies.

### 4.5.1 Research Questions

Our evaluation aims to answer the following research questions (RQs):

**RQ1. How accurate is our approach at extracting glossary terms?** A set of candidate terms is accurate if it neither includes too many unwanted terms (false positives) nor misses too many desired terms (false negatives). The aim of RQ1 is to evaluate the accuracy of text chunking, enhanced with our heuristics, at detecting glossary terms.

**RQ2. Which similarity measure(s) and clustering algorithms(s) yield the most accurate clusters?** The choice of similarity measures and clustering algorithm can have a major impact on the quality of the generated clusters. The aim of RQ2 is to examine alternative combinations of similarity measures and clustering algorithms, and identify the best alternatives in terms of accuracy.

**RQ3. How can one specify the number of clusters?** A bad choice for the number of clusters can compromise the accuracy of clustering and potentially render the resulting clusters useless. The aim of RQ3 is to develop systematic guidelines for choosing an appropriate number of clusters for a specific requirements document.

**RQ4. Which of the alternatives identified in RQ2 are the most accurate when used with the guidelines from RQ3?** RQ2 uses an averaging metric for identifying the most accurate clustering algorithms and similarity measures. This metric is *not* a direct indication of accuracy at a fixed number of clusters. From a practical standpoint, one needs to know which combinations of clustering algorithms and similarity measures are best when the number of clusters is set as per the recommendation from RQ3. The aim of RQ4 is to find the combinations that work best with the guidelines of RQ3.

**RQ5. Does our approach run in practical time?** One should be able to perform candidate term

extraction and clustering reasonably quickly, even when faced with a large number of requirements. The aim of RQ5 is to investigate whether our approach has a practical running time.

**RQ6. How effective is our clustering technique at grouping related terms?** Clustering can be a useful assistance to analysts during glossary construction only if the generated clusters are sufficiently accurate. Drawing on the clustering accuracy results from our case studies, RQ6 argues about the overall effectiveness of our clustering technique.

**RQ7. Do practitioners find the clusters generated by our approach useful?** Ultimately, our clustering technique is valuable only if practitioners faced with real Requirements Engineering tasks find the generated clusters useful. RQ7 is aimed at assessing the perceptions of the experts involved in our case studies about the usefulness of the generated clusters.

## 4.5.2 Description of Case Studies

Table 4.6 provides, for each of our case studies, a short description, the case study domain, and the number of requirements statements in the respective requirements document.

The first case study, hereafter *Case-A*, concerns a software component developed by SES Techcom – a satellite communication company – for a satellite ground station. *Case-A* has 380 requirements. The second case study, hereafter *Case-B*, concerns a safety evidence management system built in an EU project, OPENCOSS (<http://www.opencoss-project.eu>), with participation from 11 companies and 4 research institutes. *Case-B* has 110 requirements. The third case study, hereafter *Case-C*, concerns a satellite data dissemination network developed in a European Space Agency (ESA) project with participation from several telecommunication companies. *Case-C* has 138 requirements. *Case-A* and *Case-C* are proprietary; whereas *Case-B* is public. To facilitate replication, we make the material for *Case-B* available on our tool’s website (see Section 4.4).

For each case study, we involve a subject matter expert with in-depth knowledge about the respective case. In *Case-A* and *Case-B*, the experts were requirements analysts who were closely involved in drafting the requirements; and in *Case-C*, the expert was the project manager.

We have used the requirement documents in *Case-A* and *Case-B* as case study material before [Arora et al., 2015a]. In both cases, the requirements writers had made a conscious attempt to structure the requirements sentences according to Rupp’s template [Pohl, 2010]. Specifically, 64% of the requirements in *Case-A* and 89% of the requirements in *Case-B* conform to Rupp’s template [Arora et al., 2015a]. We are using *Case-C* as case study material for the first time. No particular template was used in the requirements of *Case-C*. In Section 4.6, we argue why the use of a template in *Case-A* and *Case-B* does not pose major validity threats. Except for the elicitation of glossary terms for *Case-A* and *Case-B* (see Section 4.5.4.1), all the empirical work reported in this section was conducted as part of our current research.

## 4.5.3 Case Selection Criteria

We had the following criteria in mind when selecting our case studies:

- We were interested in requirements documents that are reasonably large ( $> 100$  requirements), first, because automated term extraction and clustering is unlikely to provide compelling benefits over very small requirements documents, and second, because we would not be able to adequately evaluate the execution time of our approach (RQ5) using small documents.
- We wanted to cover cases from different domains. In general, conducting multiple case studies is useful for mitigating external validity threats. In our investigation, increasing external validity is particularly important for RQ3, due to the impact that the choice of the number of clusters has on the quality of the generated clusters.
- We wanted to work on cases where we could have direct access to subject matter experts. Particularly, to evaluate the accuracy of our approach, we need a gold standard, covering both the ideal glossary terms and the ideal clusters of related terms. Building a trustworthy gold standard requires deep knowledge about the problem domain and a significant level of commitment. Consequently, ensuring the availability of experts throughout our investigation was an important criteria.
- We were interested in requirements from recent or ongoing projects. Old requirements are unsuitable for our evaluation, both due to the experts' potential lack of interest to revisit these requirements, and also due to the high likelihood that the experts would not be able to readily remember all the details.

The cases we have selected satisfy the above criteria.

#### 4.5.4 Data Collection Procedure

Data collection was targeted at building the ideal set of glossary terms and clusters. We elicited the ideal glossary terms directly from the experts. As for the ideal clusters, they were elicited indirectly and through the construction of a domain model. Below, we detail the process for glossary term elicitation and domain model construction. The process for deriving ideal clusters from a domain model is discussed as part of our analysis procedure (see Section 4.5.5.1). Note that the domain model and ideal clusters are *only for evaluation purposes* and not a prerequisite for applying our approach.

##### 4.5.4.1 Glossary Term Elicitation

Despite the requirements in all our case studies having reached stability, no glossary was available for the requirements yet. To identify the glossary terms, we held walkthrough sessions with the respective expert in each case study. In these sessions, the expert would first read an individual requirements statement and then identify the glossary terms in that particular statement. The expert was asked to specify all the glossary terms in a given statement, irrespective of whether the terms had been already seen in the previous statements. When the expert was doubtful as to whether a term belonged to the glossary, they were instructed to include the term rather than leave it out, as recommended by glossary construction best practices [Pohl, 2010].

The researchers' role in the walkthrough sessions was limited to moderating the sessions and keeping track of the experts' choices about the glossary terms. Once the expert in each case study reviewed all the requirements statements in the case study, a duplicate-free list of the terms chosen by the expert for the glossary was created. For Case-A and Case-B, these lists were built as part of our previous work [Arora et al., 2015a]. The experts were allowed to revise these lists during domain

model construction (described next), which took place after glossary term elicitation. The final lists of terms are used as the gold standard for answering RQ1.

#### 4.5.4.2 Domain Model Construction

To evaluate the accuracy of our clustering technique, we need a set of ideal clusters. Rather than eliciting the ideal clusters directly, we first build a domain model – a conceptual representation of a domain – and then *infer* the ideal clusters from this domain model using the procedure described later. Intuitively, we would like each ideal cluster to bring together some glossary term and its “related” terms. The role of a domain model in this context is to specify what “related” means for every glossary term.

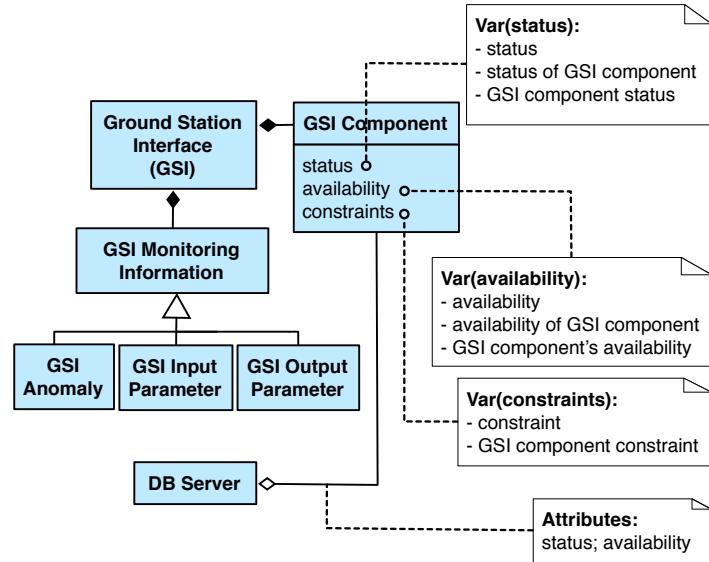
We observe that behind every requirements document, there is a domain model. This domain model may never be built explicitly, or may be partial when it is built. Nevertheless, the observation has useful implications in terms of evaluating our approach. Particularly, given a domain model and a mapping from each concept and attribute of this model onto the terms in the requirements document, one can come up with a systematic procedure for inferring the ideal clusters (Section 4.5.5.1). Such a procedure presents two key advantages: First, it alleviates the need for the domain experts to construct the ideal clusters manually – a task that is very laborious for large requirements documents such as those in our case studies. Second, although one can never entirely remove subjectivity from how a domain model is constructed and how relatedness is defined, by building an explicit domain model and formulating relatedness in a precise way, one can subject our evaluation process to scientific experimentation.

For the purposes of our evaluation, the main property we seek in a domain model is the following: Given a glossary term  $t$ , the domain model should be able to give us all the terms in the underlying requirements document that are conceptually related to  $t$ . We limit conceptual relationships to *specializations*, *aggregations* and *compositions*. Specializations represent is-a relationships. Aggregations and compositions both denote containment relationships, with the difference being that, in aggregations, the contained objects can exist independently of the container; whereas in compositions, the contained objects are owned by the container and thus cannot exist independently of it. Specializations, aggregations, and compositions constitute some of the most basic relationships between concepts and are thus instrumental for capturing relatedness between terms.

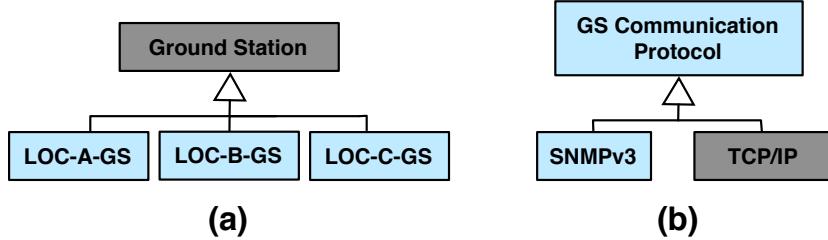
In Figure 4.7, we show a small (and sanitized) fragment of the domain model for Case-A. We use UML class diagrams for expressing domain models, as is common in object-oriented analysis [Larman, 2005]. In the figure, compositions are shown using a solid diamond shape and aggregations – using a hollow diamond shape.

We note that a domain model is not merely a structured representation of the content of a requirements document. This model further has to account for the tacit information that is not reflected in the requirements but is yet essential for properly relating the terms in the requirements. Examples of such tacit information in the class diagram of Figure 4.7 are the composition associations from `GSI` to `GSI Monitoring Information` and `GSI Component`.

We associate each element (concept or attribute)  $x$  in the domain model with a set,  $Var(x)$ , of variant terms that are conceptually equivalent to  $x$ . For example, consider the `availability` attribute of



**Figure 4.7.** A fragment of the domain model for Case-A.



**Figure 4.8.** Domain model versus glossary: grayed-out model elements have no corresponding glossary term.

GSI Component in the model fragment of Figure 4.7. This attribute is referred to in the requirements document using three variant terms: “availability” (where the link to GSI Component is implicit), “availability of GSI Component” and “GSI component’s availability”. To avoid clutter in the figure, we have not shown  $Var(x)$  when this set has only one term and that term coincides with the name label of  $x$ .

We use the notion of *Var* to describe how we built the domain models. We first reviewed the requirements document in each case study to identify all variants of the glossary terms elicited previously from the respective expert (Section 4.5.4.1).

Let  $T$  be the set of glossary terms and all variants thereof, discovered through the aforementioned review. We built our domain model  $\mathcal{M}$  for each case study in a way to ensure that all the terms in  $T$  were represented by some element  $x$  in  $\mathcal{M}$ , i.e., to ensure that  $T \subseteq \bigcup_{x \in \mathcal{M}} \text{Var}(x)$ . For our purposes, we would have liked  $\mathcal{M}$  to represent nothing but the terms in  $T$ , i.e. to have  $T = \bigcup_{x \in \mathcal{M}} \text{Var}(x)$ . However, we found this constraint to be restrictive in that it could reduce the logical completeness of the domain model. We illustrate this point using the domain model fragments (from Case-A) that are shown in Figure 4.8.

In the model fragments of Figure 4.8, the grayed-out elements, **Ground Station** and **TCP/IP**, have no corresponding terms in the glossary although both elements have corresponding terms in the requirements document. In the model fragment of Figure 4.8(a), the expert decided that the specific

<b>Statement 1.</b> I find this cluster helpful for identifying the related terms for a glossary term.
<input type="checkbox"/> Strongly Agree <input type="checkbox"/> Agree <input type="checkbox"/> Neutral <input type="checkbox"/> Disagree <input type="checkbox"/> Strongly Disagree
<b>Statement 2.</b> As the result of seeing this cluster, I can define a glossary term more precisely than I originally had in mind.
<input type="checkbox"/> Strongly Agree <input type="checkbox"/> Agree <input type="checkbox"/> Neutral <input type="checkbox"/> Disagree <input type="checkbox"/> Strongly Disagree
<b>Statement 3.</b> I find this cluster helpful for identifying the variations (synonyms) of a glossary term.
<input type="checkbox"/> Strongly Agree <input type="checkbox"/> Agree <input type="checkbox"/> Neutral <input type="checkbox"/> Disagree <input type="checkbox"/> Strongly Disagree <input type="checkbox"/> Not Relevant

**Figure 4.9.** Statements for assessing the usefulness of a cluster.

ground stations built at locations A, B, and C (actual locations sanitized) would need to be defined in the glossary; whereas, the abstract concept of ground station would not. A similar situation applies to the model fragment of Figure 4.8(b): although it is important, for completeness reasons, to model TCP/IP as a protocol alongside SNMPv3, the expert did not see a need to define TCP/IP in the glossary because it is a widely-known and standard protocol.

In general, we attempted to closely orient the domain models in our case studies around the glossary terms. Nevertheless, in situations like those illustrated in Figure 4.8, we opted to keep the non-glossary-related elements in the domain model for completeness. The domain models were built collaboratively with the involved experts. As noted in Section 4.5.4.1, the experts were allowed to refine their choice of glossary terms based on insights gained during domain model construction.

Finally, to be able to properly handle requirements about data storage and transfer, we made a modeling decision that we illustrate using requirements R2 and R6 of Figure 4.1(a). These requirements envisage that “DB server” shall store the status and availability of “GSI Component”. As shown in Figure 4.7, we model the relationship between DB Server and GSI Component as an aggregation, while keeping track of any specifically-named attributes, here, status and availability, that participate in the relationship. For deriving ideal clusters from such an aggregation, as we explain in Section 4.5.5.1, we use the participating attributes rather than the contained concept itself. This strategy helps make the ideal clusters more precise and better aligned with the requirements document.

#### 4.5.4.3 Expert Interview Survey

We conducted an interview survey with the experts in our case studies in order to assess the experts’ perceptions about the usefulness of our approach. Specifically, we chose a subset of the generated clusters in each case study, and asked the expert in that case study to evaluate these clusters individually on the basis of the three statements shown in Figure 4.9. For Case-A and Case-C, the clusters in the survey are a random selection of 20 from our tool’s output when executed with the default settings presented in Section 4.4. For Case-B, the tool yields 27 clusters, all of which are covered in the survey.

The statements in Figure 4.9 address three important tasks that analysts need to perform during the construction of a glossary: Statement 1 concerns the identification of related terms. Statement 2 concerns writing definitions for the glossary terms. The rationale for including Statement 2 in our survey is that the additional context provided by a cluster (when compared to disparate individual terms) can help the analysts in writing more precise definitions for the glossary terms.

Statement 3 addresses a specific type of related terms, namely variations (synonyms). Although related terms are already covered by Statement 1, we elected to have a dedicated statement about variations, since variations can potentially be undesirable: from our experience, we observe that industrial requirements are prone to containing *unintended* variations, both due to the flexibility of natural language and also due to differences in terminology and style across different individuals and organizations. It is important to bring such variations to the attention of the analysts, so that they can take appropriate action. Statement 3 specifically examines whether the generated clusters are good means for identifying variations, which in many cases are unintended and potentially unknown.

The survey for each case study was conducted in a single session. To avoid interviewee fatigue, we limited the sessions to a maximum duration of 1 hour each. At the beginning of a session, we introduced to the (respective) expert the statements in Figure 4.9 along with examples clarifying the motivation behind each statement. The relationship between Statement 1 and Statement 3 was further highlighted to the expert.

In the next step, the expert was asked to review the selected clusters in succession, and for each cluster, express their opinion about Statements 1, 2, and 3 on a five-point Likert scale [Likert, 1932]. For Statement 3, since not all clusters necessarily contain variations, the expert had an additional choice, “Not Relevant”, to be used when they believed that a certain cluster did not contain any variations. The expert was reminded that, for all three statements, the opinion should be based on the glossary terms they saw in the cluster being reviewed. In particular, the expert was told that, if they did not see any glossary terms in a cluster, they should refrain from choosing either “Strongly Agree” or “Agree”, although they may still see some benefit in the information provided by the cluster.

To ensure that the experts had a correct understanding of the statements in Figure 4.9, we asked each expert to verbalize their reasoning for the first five clusters that they reviewed.

We note that we conduct one interview per case study. Ideally, one should have interviews with multiple experts in each case to enable comparison and increase the reliability of the results. In our work, having more than one interview for each case study was infeasible because any respondent would have to have full familiarity with the requirements before they could meaningfully answer the interview questions. To mitigate the effect of potential expert errors, our interview covers a reasonably large number of clusters (at least 20 clusters, as discussed earlier) for each case study.

## 4.5.5 Analysis Procedure

### 4.5.5.1 Inferring Ideal Clusters

Equipped with a domain model  $\mathcal{M}$  and a function  $Var(x)$  for every concept and attribute  $x \in \mathcal{M}$  (as defined in Section 4.5.4), we infer the ideal clusters as we describe next.

Ideal clusters are created around concepts, with the concept attributes contributing to some of the clusters. For every concept  $c \in \mathcal{M}$ , we add to the set of ideal clusters one cluster,  $I$ , computed as follows: Let  $a_1, \dots, a_k$  denote  $c$ ’s attributes, and let  $c_1, \dots, c_n$  be the set of (parent) concepts that are directly or indirectly specialized by  $c$  (via specialization).

$$I = Var(c) \cup$$

$$\begin{aligned} & \bigcup_{1 \leq i \leq k} \text{Var}(a_i) \cup \\ & \bigcup_{1 \leq i \leq n} \text{Var}(c_i) \cup \\ & \bigcup \{ \text{Var}(a) \mid a \text{ is an attribute of some } c_i; 1 \leq i \leq n \} \cup \\ & \bigcup \{ \text{Var}(s) \mid s \text{ is a sibling of } c \text{ via some } c_i; 1 \leq i \leq n \}. \end{aligned}$$

For example, let  $c$  be the `GSI Anomaly` concept in Figure 4.7. We create a cluster by grouping together the following:  $c$ 's variant terms (only, “`GSI Anomaly`”); variant terms for  $c$ 's attributes (none); variant terms for  $c$ 's parents (“`GSI Monitoring Information`”) and parents' attributes (none); and variant terms for  $c$ 's siblings (“`GSI Input Parameter`” and “`GSI Output Parameter`”).

The above process captures relatedness between each concept and its attributes as well as between each concept and other concepts that immediately relate to it via the domain model's inheritance hierarchy. To deal with compositions and aggregations, we follow a separate process: let  $c_1$  and  $c_2$  denote the two ends of a composition or aggregation association. We add one cluster  $J = \text{Var}(c_1) \cup \text{Var}(c_2)$  to the set of ideal clusters for each such association. For example, consider the two composition associations in Figure 4.7. These induce the following clusters:

- {“`Ground Station Interface`”, “`GSI Monitoring Information`”} and
- {“`Ground Station Interface`”, “`GSI Component`”}.

The only exception to the above are aggregations in which explicitly-named attributes of the contained concept participate (see Section 4.5.4.2). For such aggregations, we bypass the contained concept and use the named attributes directly for the derivation of ideal clusters. For example, for the aggregation between `DB Server` and `GSI Component` in Figure 4.7, we create one cluster for each `status` and `availability`. This yields two ideal clusters: (1)  $\text{Var}(\text{DB Server}) \cup \text{Var}(\text{status})$ , and (2)  $\text{Var}(\text{DB Server}) \cup \text{Var}(\text{availability})$ .

Our treatment of composition and aggregation associations is motivated by the fact that while a container concept (e.g., `Ground Station Interface`) is related to each of the contained concepts, there is a weaker or no relationship between the contained concepts (e.g., `GSI Component` and `GSI Monitoring Information`). Hence, putting the contained concepts together into the same cluster, only because they happen to be contained by the same container concept, does not seem reasonable.

After creating the ideal clusters in the manner described above, we remove duplicates and any ideal cluster  $I$  that is properly contained in some other ideal cluster  $I'$  (i.e., if  $I \subset I'$ ). We use the resulting clusters as the gold standard for evaluation. The ovals in Figure 4.16 show the ideal clusters for the example of Figure 4.1. We discuss Figure 4.16 further when addressing RQ6 (Section 4.5.6.6).

#### 4.5.5.2 Evaluation Procedure

There are two main evaluation procedures underlying our empirical results: one is for assessing the accuracy of candidate terms, and the other – for assessing the accuracy of clusters:

**Accuracy of candidate terms.** We use standard classification accuracy metrics, *precision* and *recall* [Manning et al., 2008], to evaluate the accuracy of candidate terms (step 1 of the approach in Figure 4.3). The task at hand is to determine which extracted terms belong to the glossary and which ones do not. The extracted terms that belong to the glossary are True Positives (TP), and the ones

that do not belong are False Positives (FP). The glossary terms that are not extracted by our tool are False Negatives (FN). Precision accounts for the quality of results, i.e., smaller number of FPs, and is computed as  $TP / (TP + FP)$ . Recall accounts for the coverage, i.e., smaller number of FNs, and is computed as  $TP / (TP + FN)$ . We use  $F$ -measure [Manning et al., 2008] to combine precision and recall into one metric.  $F$ -measure is computed as:  $2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$ .

**Accuracy of clustering.** The choice of metrics for evaluating the accuracy of clustering (step 3 of the approach in Figure 4.3) is not as straightforward. A wide range of metrics exist to this end, each with its own advantages and limitations. Clusters are typically evaluated across two dimensions: homogeneity and completeness [Rosenberg and Hirschberg, 2007]. Homogeneity captures the intuition that the data points (in our case, candidate terms) in a generated cluster should be originating from a single class (i.e., a single ideal cluster). Completeness captures the intuition that all the data points in a class (i.e., an ideal cluster) should be grouped together in one generated cluster. A common limitation of several clustering evaluation metrics, e.g., information-theoretic measures such as entropy, is that they are not particularly suited for situations where the clusters are overlapping [Amigó et al., 2009].

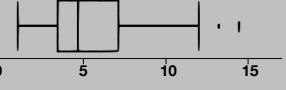
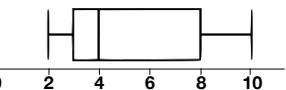
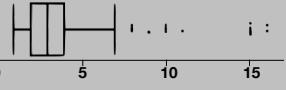
In our work, meaningful handling of overlaps is essential: while our clustering approach (Section 4.3.3) produces partitions, i.e., non-overlapping clusters, our ideal clusters (Section 4.5.5.1) are overlapping. To evaluate the accuracy of clustering, we use a simple set of accuracy metrics – a standard generalization of precision, recall and  $F$ -measure for clusters [Zhao et al., 2002] – which is straightforward to interpret in the presence of overlaps. Below, we outline the procedure for calculating these accuracy metrics for clusters. We discuss the implications of using partitioning clustering in RQ5; see Section 4.5.6.5.

Let  $I_1, \dots, I_t$  denote the set of *ideal* clusters, and let  $G_1, \dots, G_u$  denote the set of *generated* clusters.

- For every pair  $(I_i, G_j)$   $1 \leq i \leq t; 1 \leq j \leq u$ :
  - Let  $n_{ij}$  be the number of common data points between  $I_i$  and  $G_j$ .
  - $\text{Precision}(I_i, G_j) = \frac{n_{ij}}{|G_j|}$ .
  - $\text{Recall}(I_i, G_j) = \frac{n_{ij}}{|I_i|}$ .
  - $F\text{-measure}(I_i, G_j) = \frac{2 \times \text{Precision}(I_i, G_j) \times \text{Recall}(I_i, G_j)}{\text{Precision}(I_i, G_j) + \text{Recall}(I_i, G_j)}$ .
- For every ideal cluster  $I_i$   $1 \leq i \leq t$ :
  - Let  $G_r$  be the best match for  $I_i$  among generated clusters, i.e.,  $F\text{-measure}(I_i, G_r) \geq F\text{-measure}(I_i, G_j)$  for any  $1 \leq j \leq u$ . Let  $P_{\text{best\_match}}(i)$  denote  $\text{Precision}(I_i, G_r)$  and let  $R_{\text{best\_match}}(i)$  denote  $\text{Recall}(I_i, G_r)$ .
- Let  $n = \sum_{i=1}^t |I_i|$ .
- Compute overall precision and recall as weighted-averages of the precisions and recalls of the ideal clusters:
  - $\text{Precision} = \sum_{i=1}^t \frac{|I_i|}{n} \times P_{\text{best\_match}}(i)$ .
  - $\text{Recall} = \sum_{i=1}^t \frac{|I_i|}{n} \times R_{\text{best\_match}}(i)$ .

$F$ -measure for clustering is computed as the harmonic means of *Precision* and *Recall* above. This generalization of classification accuracy metrics for clusters is the basis for answering RQ2, RQ4 and RQ6 in Section 4.5.6.

Table 4.5. Information about the case studies.

Case	No. of glossary terms	No. of elements in domain model		No. of ideal clusters	Size distribution of ideal clusters	No. of generated clusters covered in interview survey
Case-A	140	Concepts (Classes)	238	119		20
		Attributes *	37			
		Specializations	143			
		Associations	58			
Case-B	51	Concepts (Classes)	35	29		27
		Attributes *	14			
		Specializations	5			
		Associations	29			
Case-C	200	Concepts (Classes)	274	142		20
		Attributes *	35			
		Specializations	110			
		Associations	123			

\* The number of domain model attributes in all three case studies is proportionally small. The reason is that, in line with best practices, when there was uncertainty as to whether an element should be a concept (class) or an attribute, we modeled it as a concept.

## 4.5.6 Results and Discussion

In this section, we describe the results of our case studies and answer the RQs stated in Section 4.5.1.

Table 4.5 provides overall statistics about the outcomes of data collection, showing, for each case study, the number of elicited glossary terms, the number of elements in the developed domain model, the number and size distribution of ideal clusters, and the number of clusters reviewed by the respective expert in the interview survey. For Case-A, a domain model existed beforehand. The researchers elaborated this model to achieve the desired characteristics detailed in Section 4.5.4.2. In Case-B and Case-C, no domain model existed a priori. The researchers built a domain model in each of these two cases by following standard practices for domain modeling [Larman, 2005], and in a way as to ensure the desired characteristics. In all three case studies, the resulting domain model was thoroughly validated with the involved expert.

### 4.5.6.1 RQ1. How accurate is our approach at extracting glossary terms?

Our terms extraction tool, REGICE, yielded 604 candidate terms for Case-A, 91 terms for Case-B, and 630 terms for Case-C. Figure 4.10 shows the classification accuracy results for our term extraction technique and compares them against the results from five existing term extraction tools, outlined in Section 4.2.4.1. We note that one of these tools, JATE, implements several alternative term extraction techniques. The results in the chart of Figure 4.10 are for the technique by Frantzi et al. [Frantzi et al., 2000], which, among the alternatives in JATE, has the best accuracy over our case studies.

As the chart indicates, our term extraction technique has better recall than the existing tools considered. Furthermore, and in terms of precision, our technique yields better results than all but TextRank in Case-A and Case-C. In both of these cases, the precision loss compared to TextRank is small (0.9% in Case-A and 1.2% in Case-C); whereas the gain in recall is large (23.6% in Case-A and 29.5% in Case-C). In other words, when compared to TextRank, our approach produces a small

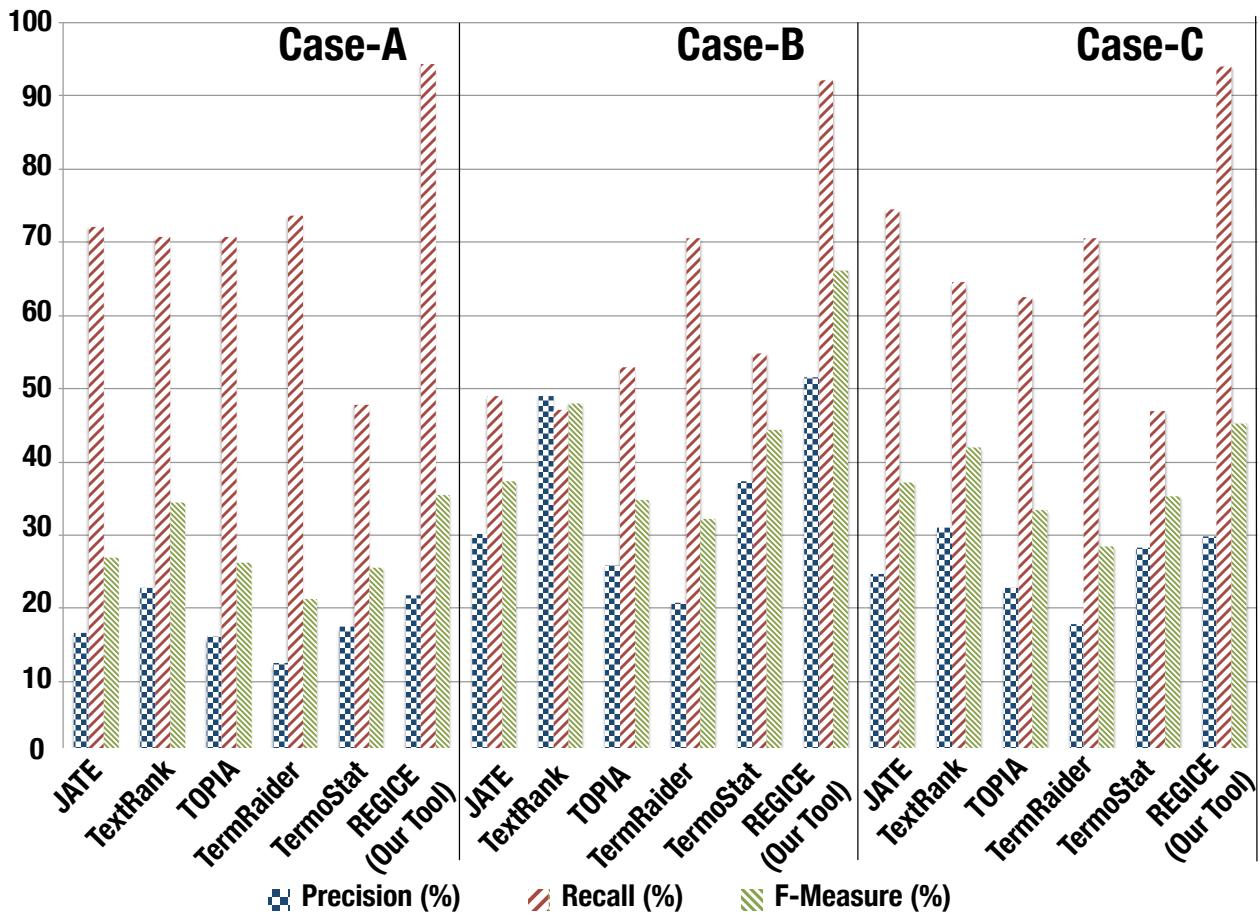


Figure 4.10. Accuracy of terms extraction.

number of additional unwanted terms (false positives) in Case-A and Case-C; but, at the same time, our approach identifies a large number of desirable terms that TextRank misses.

For glossary term extraction, recall is a more important factor than precision. A low recall (i.e., a high number of false negatives) means that the analysts will miss many of the terms that should be included in the glossary. Low precision is comparatively easier to address, as it entails only the removal of undesired items (false positives) from a list of extracted candidate terms. Given that our term extraction technique offers recall improvements of 20% or more over existing state-of-the-art tools, and at the same time, maintains or improves precision, it is reasonable to conclude that our technique is advantageous for the task of extracting glossary terms from requirements documents.

Our technique yields 8 false negatives in Case-A, 4 false negatives in Case-B, and 12 false negatives in Case-C. Of these 24 false negatives in total, 13 are explained by the heuristic we apply for filtering single-word common nouns (Table 4.4). From the remaining 11 false negatives, 4 are explained by our decision not to include VPs as candidate terms. The other 7 are due to limitations in the underlying NLP technology (i.e., mistakes made by the text chunking pipeline). Including single-word common nouns and VPs would address 17 out of the 24 false negatives. However, doing so would have a substantial and non-negligible negative impact on precision by introducing many additional false positives (precisely, 773 new false positives across the three case studies).

We further observe that VPs account for only 0.01% of the glossary terms in our case studies. This is consistent with the findings of Justeson and Katz [Justeson and Katz, 1995] and their conclusion that VPs contribute  $\leq 1\%$  of the terms in technical glossaries.

The results in the chart of Figure 4.10 prompted an investigation as to why precision for Case-A and Case-C is lower across all techniques, including ours. To identify the cause, we asked the experts in Case-A and Case-C to explain the rationale in choosing the ideal glossary terms. We determined that their choices exclusively reflected the terms for the glossary of the specific requirements documents being analyzed. In other words, terms that were deemed common knowledge or were known to be already defined in the glossaries of related documents were excluded. In general, since such contextual factors and working assumptions are often tacit and thus unavailable to an automated tool, large variations may be seen in terms of precision across different projects. Nevertheless, recall, which is the primary factor for glossary term extraction as we argued above, will not be affected.

#### **4.5.6.2 RQ2. Which similarity measure(s) and clustering algorithm(s) yield the most accurate clusters?**

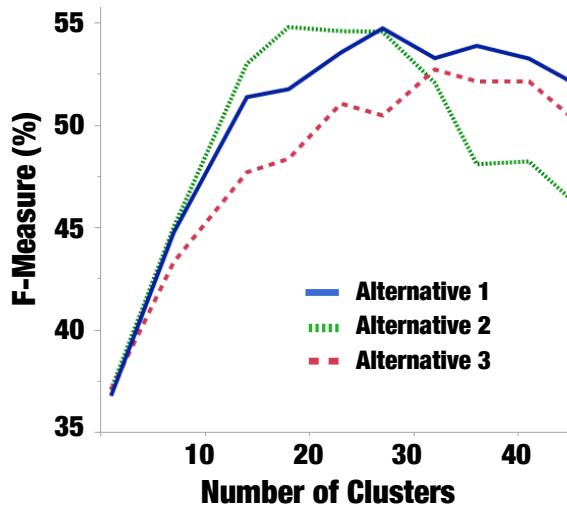
We assess the accuracy of clustering based on the  $F$ -measure metric for clusters, defined in Section 4.5.5.2. As suggested by the discussion in RQ1, the set of candidate terms in a requirements document can be wider than the ones that are of interest for glossary construction. However, the generated clusters cover *all* candidate terms, not only those that are relevant to the glossary. To determine which similarity measure(s) and clustering algorithm(s) produce the best results relevant to the glossary, we need to discard in our analysis of accuracy the terms that are not relevant. Specifically, for the purpose of  $F$ -measure calculation, we prune from the generated clusters any term that is not in at least one of the ideal clusters.

We note that outside an evaluation setting, one cannot distinguish terms that are relevant to the glossary from those that are not. To preserve the realistic behavior of clustering, it is thus paramount to compute the generated clusters for all the candidate terms first and then prune the results for evaluation, as opposed to narrowing the set of candidate terms to those that are relevant and then clustering only the relevant terms.

To answer RQ2, we considered pairwise combinations of the 12 syntactic and 8 semantic similarity measures introduced in Section 4.2.2. Three semantic measures, *HSO*, *LESK*, and *LCH*, were discarded during initial analysis due to scalability issues. The total number of remaining combinations is  $(12 + 1) \times (5 + 1) - 1 = 77$ . These combinations include configurations where an individual syntactic or semantic measure is applied on its own.

For clustering, we considered *K-means*, *Hierarchical* and *EM*, as introduced in Section 4.2.3.1. We experimented with 8 variant cluster distance functions (Table 4.3) for hierarchical clustering. This brings the total number of clustering algorithms to 10. We evaluated each clustering algorithm in conjunction with all the 77 possible combinations of similarity measures, i.e., a total of  $10 \times 77 = 770$  alternative cluster computations per case study.

For a given cluster computation alternative, i.e., combination of a clustering algorithm and similarity measures, plotting  $F$ -measure against the number of clusters results in curves similar to those shown in Figure 4.11.



**Figure 4.11.** *F*-measure curves for three different cluster computation alternatives.

The shape of these curves is explained as follows: When the selected number of clusters,  $K$ , on the  $x$ -axis is small, the size of the generated clusters is large, since the average size of clusters is inversely related to  $K$ . These large-sized clusters yield high recall but low precision, with an overall low  $F$ -measure. As  $K$  increases, large clusters become smaller and more cohesive. This brings about major increases in precision without a significant negative impact on recall, thus increasing  $F$ -measure. The  $F$ -measure peaks at some  $K$  value. This is where the generated clusters are closest to the ideal ones. Beyond this optimal  $K$  (the estimation of which is the subject of RQ3), increases in  $K$  will reduce  $F$ -measure, as recall begins to drop rapidly and losses in recall are no longer offset but gains in precision.

For a set of candidate glossary terms with a cardinality of  $n$ , increasing the value of  $K$  beyond  $n/2$  would have little practical meaning, as the average size of clusters would fall below 2 terms per cluster. Generating clusters that are this small would defeat the purpose of clustering. To use curves similar to those in Figure 4.11 as an evaluation instrument, we therefore restrict the upper range for  $K$  to a maximum of  $n/2$  (upper bound).

For each of the 770 alternatives in a given case study, we plot an  $F$ -measure curve for 10 points on the  $x$ -axis at regular intervals, ranging from 1 to  $n/2$ . Note that  $n$  is the number of candidate terms in the case study in question. The reason we limit ourselves to 10 observation points is that a thorough analysis for all possible numbers of clusters is extremely expensive computationally.<sup>1</sup> Knowing already that the  $F$ -measure curves follow a certain shape, as we explained earlier, and in light of the fact that the analysis we perform over these curves, as will become clearer over the course of our discussion, is a preliminary step for identifying the most promising alternatives, we deemed 10 observation points to be sufficient for approximating the curves.

We compare the curves by taking the average of  $F$ -measures over the entire value range for  $K$ . The rationale for averaging is that, in lieu of knowledge about the optimal  $K$ , we would like to favor

<sup>1</sup>Such an analysis would have required us to execute clustering  $770 \times n/2$  times per case study, i.e.,  $770 \times (604/2 + [91/2] + 630/2) = 510510$  times in total. Our approximate estimation of the execution time for such an experiment is more than 60 days on a conventional computer.

**Table 4.6.** Top-5 cluster computation alternatives.

Case	Syntactic Measure	Semantic Measure	Clustering Algorithm	Normalized AUC
<b>Case-A</b>	Levenstein	JCN	EM	0.485
	Levenstein	PATH	EM	0.484
	Levenstein	LIN	EM	0.482
	Levenstein	PATH	K-means	0.476
	Levenstein	RES	EM	0.476
<b>Case-B</b>	SoftTFIDF	NONE	EM	0.523
	SoftTFIDF	NONE	K-means	0.519
	Jaccard	NONE	EM	0.498
	Monge_Elkan	JCN	EM	0.498
	Euclidean	NONE	EM	0.493
<b>Case-C</b>	Levenstein	LIN	EM	0.559
	Levenstein	RES	EM	0.557
	Levenstein	PATH	EM	0.557
	Levenstein	JCN	EM	0.553
	SoftTFIDF	LIN	EM	0.547

alternatives that yield the best overall accuracy across all  $K$  values. Naturally, an alternative fares better than another if it has a higher average  $F$ -measure. We compute the average  $F$ -measure for each curve by computing its *Area Under the Curve* (AUC) and normalizing the result.

In Table 4.6, we show for each case study the top-5 alternatives that yield the best average accuracy, i.e., alternatives with the largest (normalized) AUC. As suggested by the table, there is no alternative that is shared among all three case studies. We therefore cannot recommend a best alternative based on the information in this table alone.

To provide a general recommendation, we need to further consider the effect of individual syntactic measures, semantic measures and clustering algorithms on accuracy across all the alternatives. For example, a syntactic measure that is not employed in the absolute-best alternative but performs consistently well in all the alternatives where it is employed may be advantageous over one that is employed in the absolute-best alternative but also in some poor alternatives.

More precisely, we want to find similarity measures and clustering algorithms that yield good (but not necessarily the absolute-best) results, and at the same time, cause little variation. A standard way for performing such analysis is by building a regression tree [Breiman et al., 1984]. A regression tree is a step-wise partitioning of a set of data points with the goal of minimizing, with respect to a certain metric, variation within partitions. Here, the data points are the 770 (cluster computation) alternatives and the metric of interest is the AUC. For each case study, the regression tree identifies at any level in the tree the most influential factor that explains the variation between the data points. In our context, there are three factors: (1) the syntactic measure, (2) the semantic measure, and (3) the clustering algorithm. Once the most influential factor is identified, the regression tree partitions the data points into two sets in a way as to minimize variations within the resulting sets.

Figure 4.12 shows the regression trees for our case studies. In each node of the tree, we show the count (number of alternatives), and the mean and standard deviation for AUC. By convention, sibling nodes in the tree are ordered from left to right based on their mean values. That is, the node on a right branch has a larger mean than its sibling on the left. For every expanded (i.e., non-leaf) node, the node

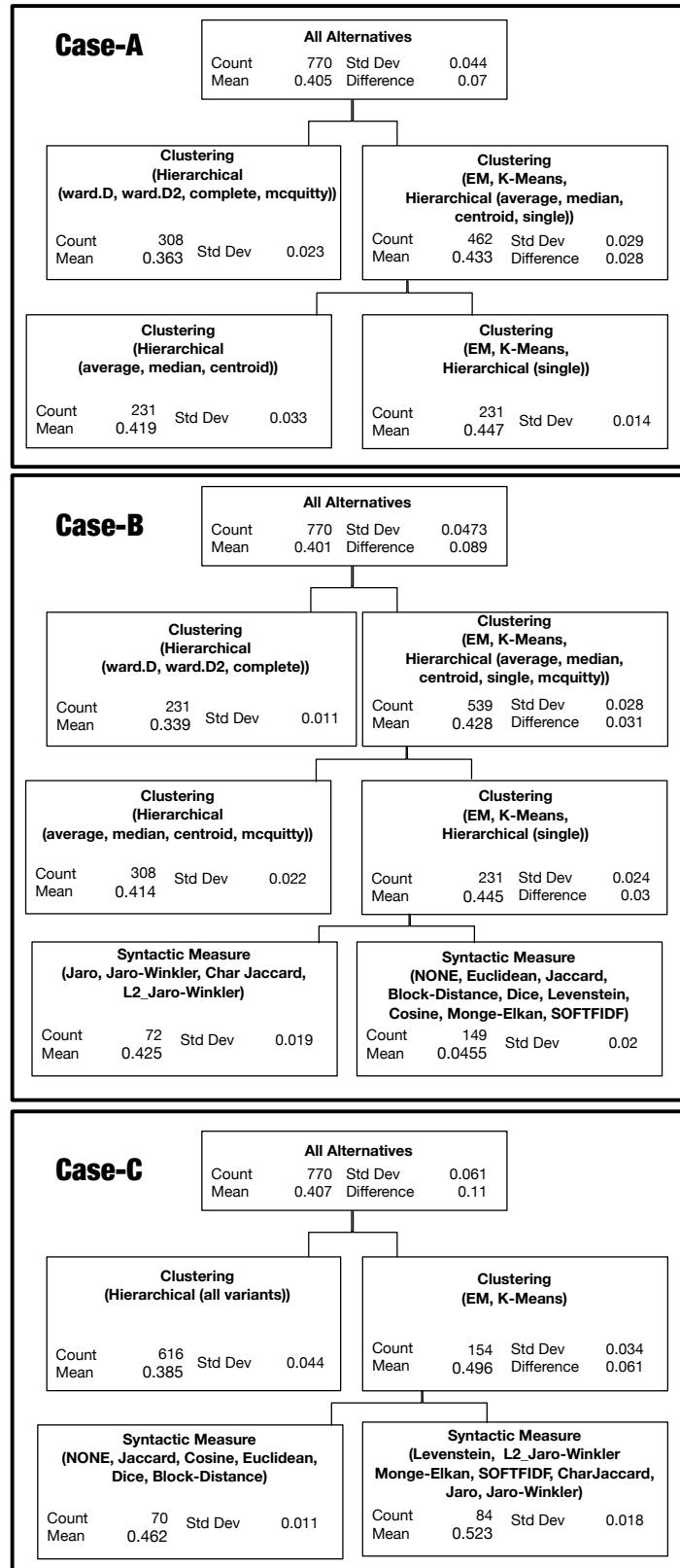
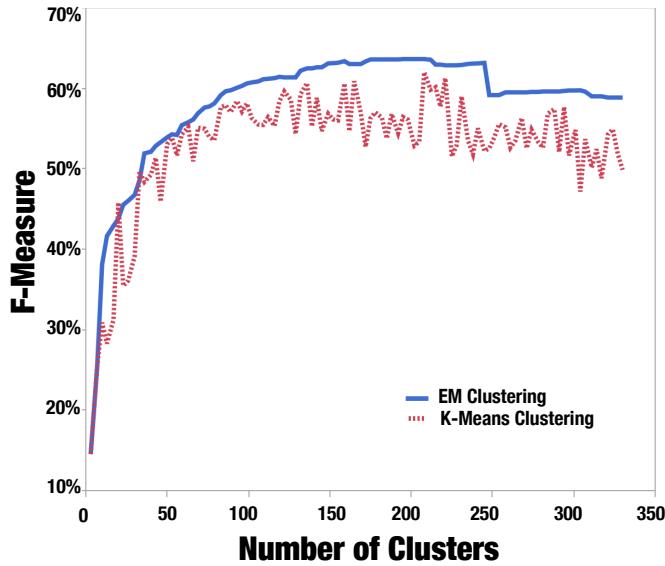


Figure 4.12. Regression trees for normalized AUCs.

shows the difference between the mean values of its right and left children. For each case study, we iteratively expanded the regression tree until the difference (in means) between the right and the left



**Figure 4.13.** Comparison between the K-means and EM clustering algorithms; both curves are for Case-C and computed using the combination of Levenshtein and PATH.

nodes fell below the threshold of 0.025, which we deemed to be the minimum difference of practical relevance. Naturally, we expanded only the right branches, given the convention we noted above.

As the regression trees of Figure 4.12 suggest, the most influential factor in all three case studies is the choice of clustering algorithm. Only EM and K-means perform consistently well across all our case studies, thus ruling out all variants of hierarchical clustering. For Case-A, the choice of similarity measures does not play a significant role. But, for Case-B and Case-C, the choice of syntactic measure is the second most influential factor, as shown by the regression trees of these two case studies. Taking the overlap between the best-performing syntactic measures in Case-B and Case-C, we narrow the choices of syntactic measures to Levenshtein, Monge-Elkan and SoftTFIDF. The label NONE, appearing in the regression trees of Case-B and Case-C, denotes stand-alone applications of *semantic* measures (i.e., without a syntactic measure). We observe that in Case-C, NONE appears on a left branch. We thus conclude that individual semantic measures should *not* be applied on their own.

Our regression tree analysis finds both EM and K-means to be good choices for the clustering algorithm. To develop further insights into the behavior of EM and K-means, we plotted higher-resolution *F*-measure curves for both algorithms when applied in combination with the best similarity measures identified through our regression tree analysis. Specifically, we increased the number of points on the *x*-axis from 10 (used for the charts of Figure 4.11) to 100.

Figure 4.13 illustrates the higher-resolution *F*-measure curves. The two curves in this figure differ only in the choice of the clustering algorithm. We observe from the figure that the curve for EM is relatively smooth; whereas the one for K-means has spikes, indicating that K-means is very sensitive to the selected number of clusters. This trend of behavior was seen across all our case studies and all the similarity measures considered. The high sensitivity of K-means to the number of clusters is undesirable in our context, where we can never exactly pinpoint the optimal number of clusters and

can only provide guidelines for coming up with a reasonable estimate (RQ3). Consequently, we rule out K-means, thus narrowing the choice of clustering algorithm to EM.

To summarize our discussion of RQ2, we conclude that the following similarity measures and clustering algorithm are good choices for further examination in RQ4:

**Syntactic measures:** One of the three syntactic measures: *Levenstein*, *Monge-Elkan* or *SoftTFIDF*.

**Semantic measures:** Semantic measures do not have a significant impact on clustering accuracy.

**Clustering algorithm:** *EM*.

Despite semantic measures not being influential in improving clustering accuracy, we do not discourage the use of these measures. A potential reason why we did not find semantic measures useful is that semantic synonyms are not prevalent in our case studies. Semantic measures, when combined with syntactic ones, may be useful if the likelihood of (semantic) synonyms being present is high. As indicated in Section 4.4, our tool already supports semantic measures.

#### 4.5.6.3 RQ3. How can one specify the number of clusters?

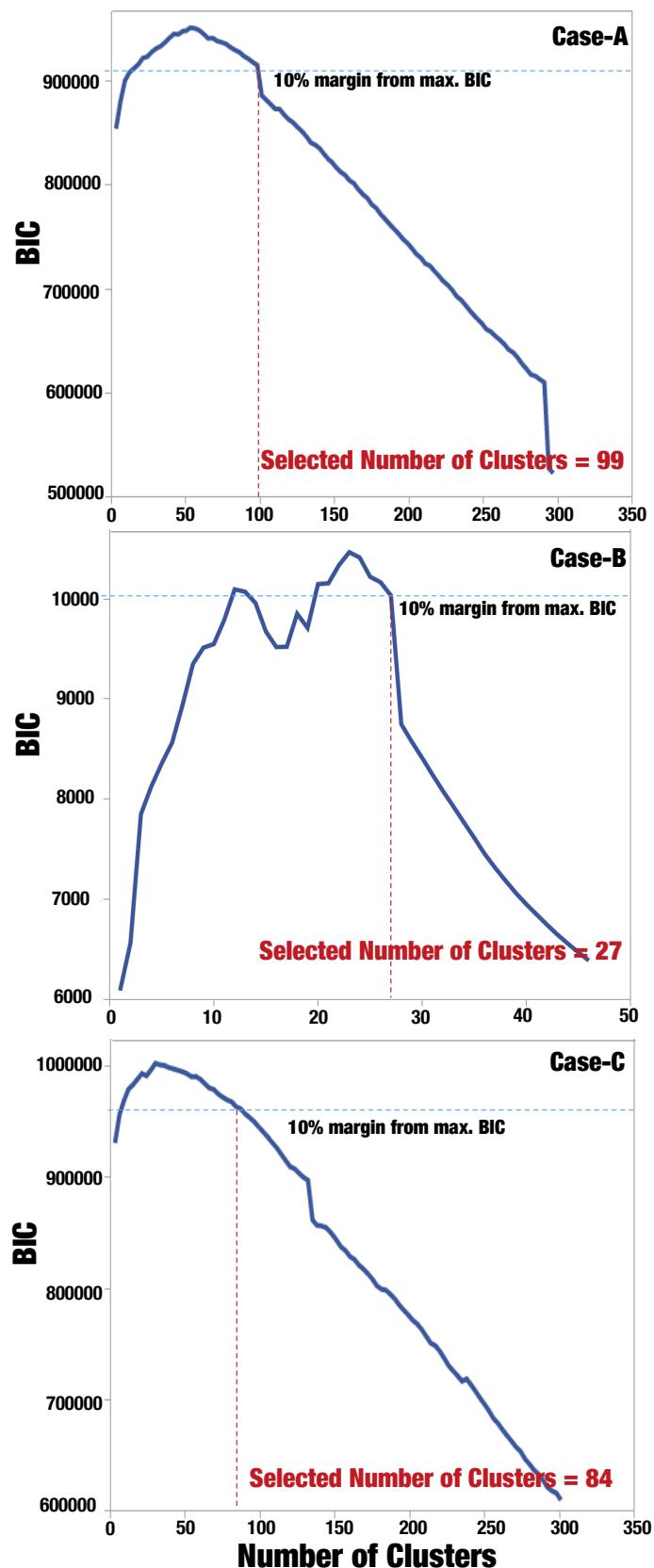
Theoretically, the number of clusters,  $K$ , is a value between 1 and the total number of candidate terms. However, as we argued in RQ2, considering a  $K$  value that is larger than half the number of candidate terms is unreasonable from a practical standpoint. If  $K$  is too small, accuracy will be low due to large clusters with low precision. If  $K$  is too large, accuracy will again be low but this time due to small clusters with low recall. Obviously, one does not have access to a gold standard outside an evaluation setting. Therefore, one cannot use the optimal point in  $F$ -measure curves such as those in Figure 4.11 for choosing  $K$ .

We use BIC, discussed in Section 4.3.3, to choose a value for  $K$ . As explained in this earlier section, the intuition is that the larger the BIC, the better is the choice for the value of  $K$ . In other words, our goal should be to choose  $K$  in a way that maximizes BIC. We apply an adaptation of this general idea for choosing  $K$ , as we describe next.

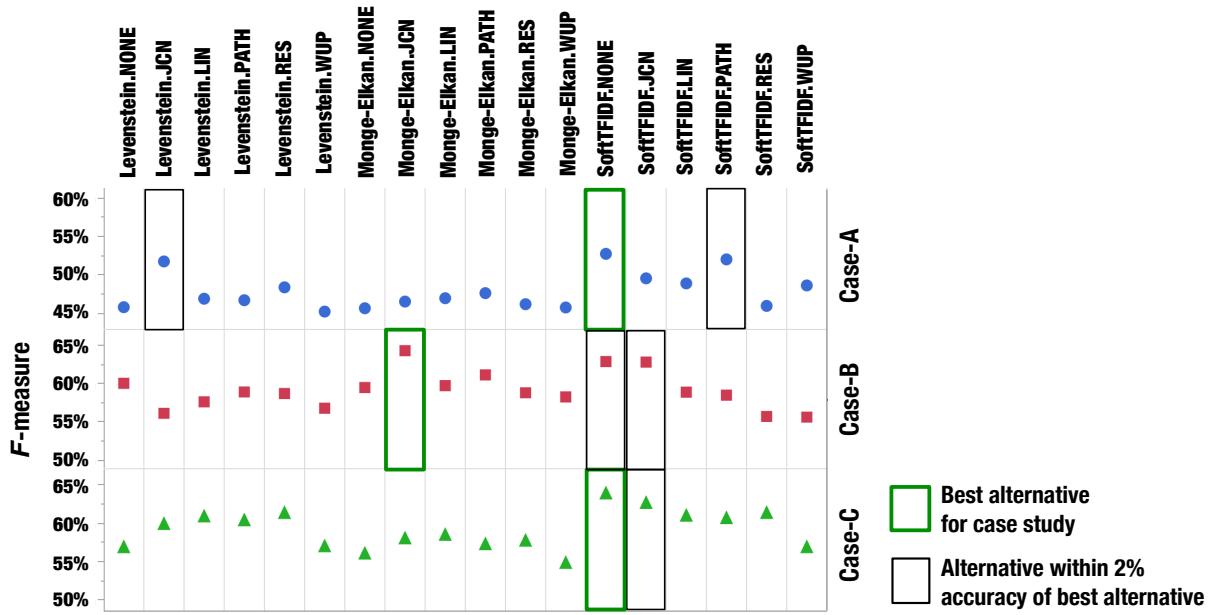
For a specific cluster computation alternative, we first plot the BIC curve over the range  $[1, n/2]$  of the number of clusters, where  $n$  is the number of candidate terms. To illustrate, Figure 4.14 shows the BIC curves resulting from the application of EM using similarities calculated with SoftTFIDF for each case study.

For performance reasons, when the number of candidate terms (and thus the number of values in the range  $[1, n/2]$ ) is large, we may elect not to compute the BIC at every value on the  $x$ -axis. Instead, like in RQ2, we may divide the  $x$ -axis into a certain number of points. For example, in Figure 4.14, the  $x$ -axis for Case-A and Case-B is divided in increments of 1% of the range (100 points). For Case-B, the range is small ( $< 100$  points); therefore, all the values in the range are covered.

Let  $BIC_{max}$  and  $BIC_{min}$  be the maximum and minimum BIC values, respectively; and let  $margin = (BIC_{max} - BIC_{min})/10$ , i.e., 10% of the difference between  $BIC_{max}$  and  $BIC_{min}$ . Assuming that  $BIC_{max}$  occurs over a peak, we look to the right of the peak and choose the largest possible  $K$



**Figure 4.14.** Selecting the number of clusters using BIC.



**Figure 4.15.** Accuracy of the alternatives identified in RQ2 when the number of clusters is set using the guidelines of RQ3.

whose BIC is larger than or equal to  $BIC_{max} - margin$ . In the (unlikely) case where the BIC curve is monotonically-increasing, i.e., there is no peak, we set  $K$  to be  $n/2$ . The rationale for choosing a  $K$  with a smaller BIC than the maximum is to obtain a larger number of clusters and hence a smaller number of terms within individual clusters. As long as BIC remains uncompromised, i.e., stays within a small margin from the maximum, clusters with a smaller number of terms are more desirable because they are more homogeneous and easier for analysts to inspect.

In the example curves of Figure 4.14,  $BIC_{max}$  for Case-A occurs when there are 55 clusters. The BIC value nonetheless stays within the 10% margin up to 99 clusters. Based on our argument above, we select  $K = 99$  for Case-A. In Case-B,  $BIC_{max}$  occurs at 23 clusters. BIC stays within the specified margin up to 27 clusters, followed by a steep decline. We therefore select  $K = 27$  for Case-B. In Case-C,  $BIC_{max}$  occurs at 30 clusters but remains within the margin up to 84 clusters. Hence, in this case, we select  $K = 84$ .

We note that our guidelines for selecting the number of clusters are *automatable* and have been already implemented into our tool support (Section 4.4). Therefore, the computation of BIC and its interpretation are transparent to the end-users of our approach.

#### 4.5.6.4 RQ4. Which of the alternatives identified in RQ2 are the most accurate when used with the guidelines from RQ3?

The analysis of RQ2 narrows cluster computation alternatives to one clustering algorithm, namely EM, and three syntactic measures, namely Levenshtein, Monge-Elkan and SoftTFIDF. The analysis further suggests that semantic measures do not influence clustering accuracy in a major way. Nevertheless, and as discussed in RQ2, we do not rule out the usefulness of semantic measures in general. For answering RQ4, we therefore consider all the viable semantic measures from RQ2, namely LIN, PATH, RES, JCN and WUP.

**Table 4.7.** Execution times.

Case / strategy to increment # of clusters	Phase	Execution Time
Case-A 380 requirements statements, 604 terms # clusters incremented by 1% of # of terms	Term Extraction	15s
	Similarity Calculation (Syntactic + Semantic)	32s + 58s = 90s
	Clustering	20m
	<b>Total</b>	21m 45s
Case-B 110 requirements statements, 91 terms # clusters incremented by 1 unit in each run	Term Extraction	72s
	Similarity Calculation (Syntactic + Semantic)	18s + 49s = 67s
	Clustering	15s
	<b>Total</b>	1m 55s
Case-C 138 requirements statements, 630 terms # clusters incremented by 1% of # of terms	Term Extraction	13s
	Similarity Calculation (Syntactic + Semantic)	33s + 61s = 94s
	Clustering	23m
	<b>Total</b>	24m 47s

Specifically, we answer RQ4 by investigating the 18 alternatives shown in Figure 4.15. We leave the clustering algorithm (EM) implicit in the figure because it is the same across all these alternatives. We represent a pairwise combination of a syntactic and a semantic measure by concatenating their names separated by the “.” symbol. For example, we write SoftTFIDF.PATH to refer to the combination of SoftTFIDF (syntactic) and PATH (semantic). When a syntactic measure is used on its own, we use NONE to indicate the absence of a semantic measure. For example, SoftTFIDF, when used alone, is denoted SoftTFIDF.NONE.

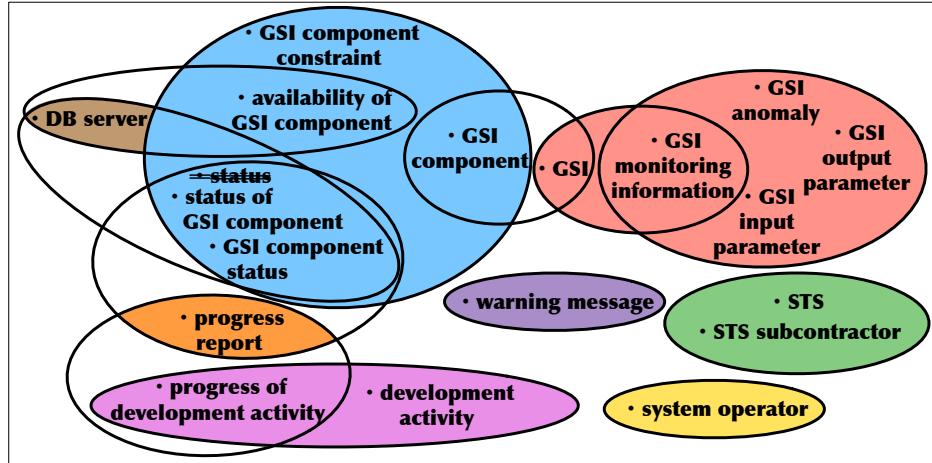
For each alternative, we first compute the BIC curve and apply the guidelines of RQ3 to select the number of clusters (according the BIC curve produced by that specific alternative). We then calculate the accuracy (*F*-measure) of the alternative at the selected number of clusters. The results are shown in Figure 4.15. For each case study, we indicate both the best alternative and also any other alternative whose accuracy is within 2% of the accuracy of the best alternative.

Our results suggest that applying SoftTFIDF alone for computing similarities between terms presents the best overall option. If the analyst elects to further use a semantic measure, the best choice would be to combine SoftTFIDF with JCN: as shown by Figure 4.15, this combination closely follows the stand-alone application of SoftTFIDF in terms of accuracy.

#### 4.5.6.5 RQ5. Does our approach run in practical time?

Table 4.7 shows the execution times for the different steps of our approach. All execution times were measured on a laptop with a 2.3 GHz Intel CPU and 8GB of memory. The execution times we report for similarity calculation in Table 4.7 are based on the combined application of SoftTFIDF and JCN, i.e., the best combination from RQ4 involving a semantic measure. This provides a more realistic picture of execution times, should the application of semantic measures be warranted.

The running time of our approach is dominated by the clustering step and more specifically by the construction of the BIC curve, outlined in Section 4.5.6.3. For calculating a BIC curve, as was discussed earlier, we consider 1% intervals on the *x*-axis, instead of every possible number of clusters. The implementation we use for generating BIC curves performs, for any point on the *x*-axis, 10 BIC

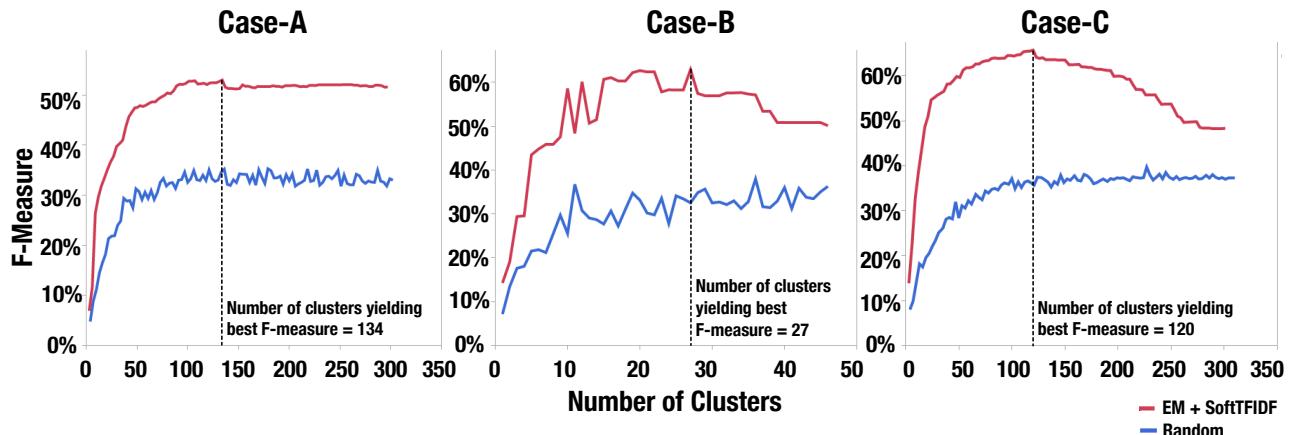


**Figure 4.16.** Clustering example: ovals represent ideal clusters and background colors represent generated clusters.

**Table 4.8.** (a) Upper bounds for the accuracy of partitioning clustering, (b) the actual accuracy of our approach.

<b>(a) Best Possible</b>			<b>(b) EM + SoftTFIDF</b>		
<b>Case</b>	<b>Precision</b>	<b>Recall</b>	<b>Absolute Precision</b>	<b>Absolute Recall</b>	<b>Absolute F-Measure</b>
Case-A	90.8%	76.2%	82.9%	66.8% (73.6%)	43.6% (57.2%)
Case-B	89.1%	70.7%	78.8%	79.7% (89.5%)	51.9% (73.4%)
Case-C	93.5%	79.1%	85.7%	77.4% (82.8%)	54.5% (68.9%)

\* Relative = Absolute / Best Possible \* 100



**Figure 4.17.** Comparison with random partitioning; the optimal number of clusters for each case study is further shown.

calculations corresponding to 10 different probabilistic models [Fraley and Raftery, 2012]. Each BIC value in the curve is the maximum of these 10 calculations.

We believe that the overall execution times in Table 4.7 are practical, noting that glossary term clustering does not need to be repeated frequently. However, handling requirements documents that are much larger than those in our case studies may require a strategy to further reduce the time spent on building BIC curves. This can be done by further limiting the number of points on the *x*-axis of these curves (e.g., by using larger intervals), or by considering only a subset of the probabilistic model alternatives.

#### 4.5.6.6 RQ6. How effective is our clustering technique at grouping related terms?

The ideal clusters in our evaluation are overlapping. The overlaps arise because individual candidate terms may assume different roles in different requirements statements, and thus relate to potentially different terms based on each role. For example, based on the model of Figure 4.7, the term “GSI monitoring information” assumes two different roles, one as a parent concept for “GSI anomaly”, “GSI input parameter”, and “GSI output parameter”, and another as a constituent part of “Ground Station Interface”. As a result, there will be two ideal clusters containing “GSI monitoring information”: one ideal cluster around “GSI monitoring information” and its child concepts (terms), and another ideal cluster around the composition relationship between “Ground Station Interface” and “GSI monitoring information”. By allowing overlaps in the ideal clusters, one can distinguish different roles and orient each ideal cluster around one particular role. Having the roles separated from one another is advantageous because it makes the clusters highly-focused and small.

The above said, generating clusters that are overlapping can pose an overhead for end-users, because individual candidate terms can appear multiple times in such clusters. This means that the effort associated with manually reviewing overlapping clusters will be, potentially by several folds, higher than the case where the generated clusters are non-overlapping. The clustering algorithms we considered in our work are partitioning algorithms. Nevertheless, we evaluated the results of these algorithms against overlapping ideal clusters to determine how close we can get to the (conceptually) ideal situation, without actually making the generated clusters overlapping.

To illustrate, we show in Figure 4.16 both the ideal clusters and the generated ones (by EM and SoftTFIDF) for the example of Figure 4.1. Here, there are 12 ideal clusters, represented as ovals, and 8 generated clusters, represented using colors. The term “status” appears in the ideal clusters but is struck out from the generated ones. This term, which is a variant of “GSI Component status” is filtered due to being a single-word common noun (see Table 4.4).

The clustering precision and recall for this example are 73.7% and 75%, respectively. These numbers mean that, on average, 73.7% of the terms an analyst sees in a generated cluster pertain to the specific aspect of relatedness they are investigating. Further, 75% of the ideal terms for this specific relatedness aspect have been retrieved. Despite the accuracy not being perfect, the generated clusters provide reasonable cues about related terms. The extent to which practitioners find the generated clusters useful is addressed in RQ7.

Given that achieving perfect accuracy is theoretically impossible (unless the ideal clusters have no overlaps), we need to find an upper bound on the maximum possible accuracy that one can expect from partitioning clustering in our context. This upper bound provides a reference point for evaluating the effectiveness of clustering in our approach.

To compute such an upper bound, we follow a randomized procedure. Specifically, we impose a random order on the ideal clusters and prune these clusters so that the following constraint holds for any given term  $t$ : if  $t$  appears in a cluster  $C$  at position  $i$  in the ordering, then  $t$  cannot appear in any cluster  $C'$  at a position  $i'$  such that  $i' > i$ . This procedure derives non-overlapping clusters from the ideal clusters. The accuracy of these non-overlapping clusters is a good indicator for what partitioning clustering can achieve in the best case. We applied the above procedure to 1000 random orders of the ideal clusters in our three case studies, and computed the average accuracy measures.

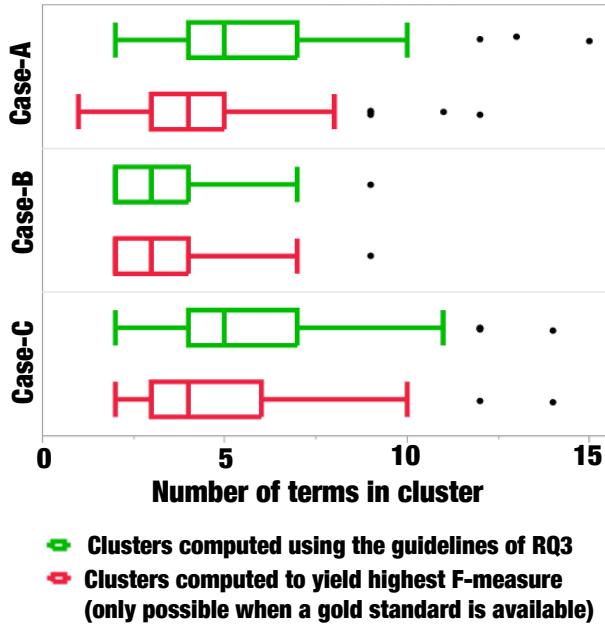


Figure 4.18. Cluster size distributions.

Table 4.8(a) shows upper bounds for precision, recall, and  $F$ -measure when the overlaps have been removed. For comparison, we provide in Table 4.8(b) the accuracy of the best alternative from RQ4 (i.e., EM + SoftTFIDF), with the number of clusters chosen as discussed in RQ3. In addition, Table 4.8(b) shows the accuracy scores relative to the upper bounds of Table 4.8(a). We believe that these results, when complemented with the expert feedback discussed later in RQ7, provide positive evidence for the effectiveness of our approach.

As a sanity check, we compare our best cluster computation alternative with random partitioning. For this purpose, we use a 10-fold random partitioning of candidate terms into equal-sized clusters. In Figure 4.17, we show the  $F$ -measure curve of the best alternative against that of random partitioning. As can be seen from the figure, the best alternative significantly outperforms the random baseline.

Figure 4.18 shows, for each case study, the size distribution of the clusters computed using the best alternative. To facilitate comparison, we show the distributions both for the situation where we select the number of clusters based on the guidelines of RQ3, and also for the situation where the number of clusters is chosen in a way as to maximize  $F$ -measure. The number of clusters maximizing  $F$ -measure in Case-A, Case-B and Case-C are 134, 27 and 120, respectively, as marked on the charts of Figure 4.17. We note that these optimal numbers of clusters are known only in an evaluation setting, i.e., when the ideal clusters are known.

The results in Figure 4.18 provide confidence that: (1) the generated clusters are reasonably small and thus easy for the analysts to review; and (2) the size distributions we obtain by following the guidelines of RQ3 are not drastically different than those obtained from an optimal (but in a realistic setting, unattainable) clustering.

A final remark we need to make about the distributions of Figure 4.18 is that these distributions are *not* directly comparable to those of the ideal clusters, shown earlier in Figure 4.5. This is because

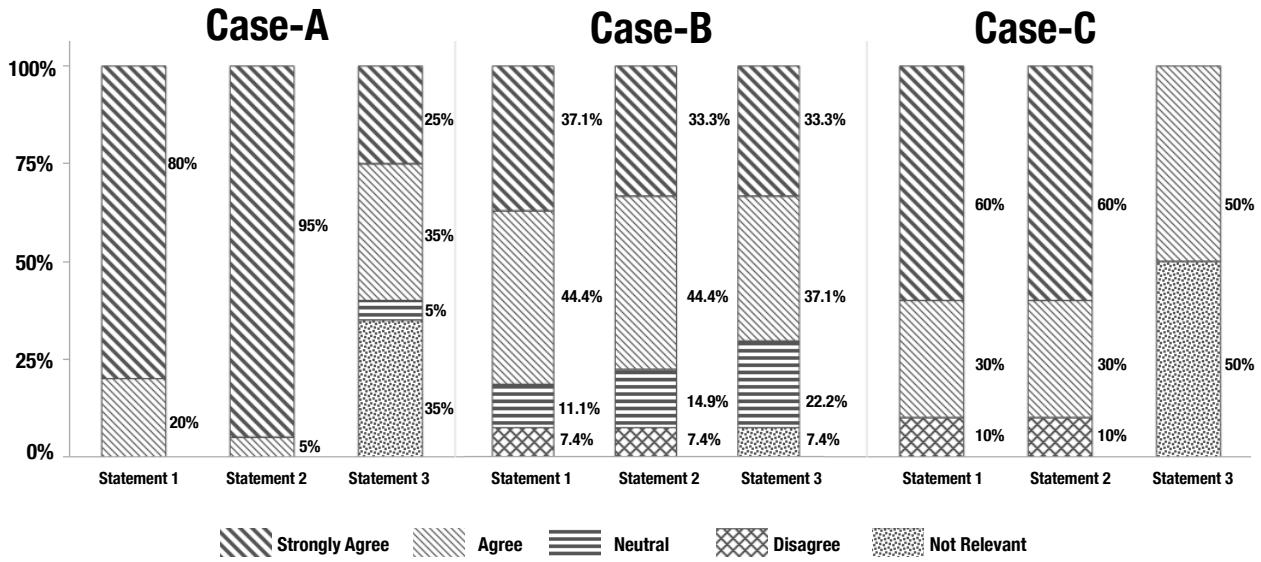


Figure 4.19. Expert survey interview results.

the generated clusters cover *all* candidate terms; whereas the ideal clusters are concerned with only the actual glossary terms.

#### 4.5.6.7 RQ7. Do practitioners find the clusters generated by our approach useful?

Figure 4.19 shows the results of our survey study, obtained by following the procedure described in Section 4.5.4.3. We recall that the number of clusters considered in our surveys for Case-A, Case-B and Case-C are 20, 27 and 20, respectively. The surveys therefore collectively examine 67 clusters.

With regard to Statement 1 (see Figure 4.9), the experts strongly agreed or agreed in 89.6% of the cases that the clusters were helpful in identifying related terms. The experts were neutral in 4.4% of the cases (3 clusters, all in Case-B), and disagreed in 6% of the cases (4 clusters, two in Case-B and two in Case-C).

With regard to Statement 2, in 88% of the cases, the experts strongly agreed or agreed that the clusters were helpful for defining the glossary terms more precisely. In 6% of the cases (4 clusters, all in Case-B), the experts were neutral; and in the remaining 6% of the cases (4 clusters, two in Case-B and two in Case-C), the experts disagreed.

Finally and with regard to Statement 3, the experts deemed 19 out of the 67 clusters (28.4%) not relevant, meaning that the experts did not see variants in these clusters. For the remaining 48 clusters (71.6%), the experts either agreed or strongly agreed that the clusters were helpful for identifying variant terms, except in 7 cases where the experts were neutral (one in Case-A and six in Case-B).

We compute the average of the expert responses by quantifying the agreement scale, from 0 for “Strongly Disagree” to 4 for “Strongly Agree”. This would give us an average of 3.40 for Statement 1, an average of 3.42 for Statement 2, and an average of 3.15 for Statement 3, noting that we exclude for Statement 3 the clusters that were deemed not relevant.

The above results indicate that the average scores for all the three statements in our survey are be-

tween “Agree” and “Strongly Agree”. This suggests that the experts had a strong positive perception of the quality of the generated clusters.

## 4.6 Threats to Validity

In this section, we discuss threats to the validity of our empirical results and the steps we have taken to mitigate these threats.

**Internal validity.** The researchers were involved in the construction of the domain models from which the ideal clusters were derived. This raises the potential problem that the domain models could be built in a way that would favor our clustering results. To mitigate bias during domain model construction, we adhered to standard guidelines for domain modeling, notably by Larman [Larman, 2005]. Further, we subjected the domain models to a thorough review and revision process with close participation from the experts, who were familiar with domain modeling but not with the exact analytical purpose of a domain model in our evaluation.

A second potential threat to internal validity is that, as we stated in Section 4.5.2, in two of our case studies, Case-A and Case-B, an attempt had been made by the requirements authors to conform to a certain template. Applying a template often leads to simpler requirements sentences. This can in turn potentially reduce the error rate of NLP, thus increasing the accuracy of term extraction over template requirements when compared to non-template requirements.

We have seen no evidence of the above phenomenon happening in practice. As we discussed in Section 4.5.6.1, the number of false negatives caused by NLP errors is very small. Of the seven such errors across the three case studies, two occurred in Case-A, one in Case-B, and four in Case-C. When normalized by the total number of extracted terms in each case study, the NLP error rate is at 0.33% (2/604) in Case-A, 1.1% (1/91) in Case-B, and 0.63% (4/630) in Case-C. These fractions, irrespective of whether templates have or have not been used, are too small to affect accuracy in a significant way. We therefore do not anticipate the use of template requirements in our evaluation to pose a major validity threat.

**Construct validity.** The definition of ideal clusters is a subjective procedure. To mitigate construct validity threats, we applied an explicit and systematic process for defining the ideal clusters, building on the notion of a domain model. This limits subjectivity in defining the ideal clusters and further makes the process repeatable.

**Conclusion validity.** The choice of clustering accuracy metrics has an impact on the conclusions drawn based on the metrics. To minimize threats to conclusion validity, we chose the metrics in a way as to best match the overlapping nature of the ideal clusters in our problem. A complementary measure for countering conclusion validity threats is the interview survey analysis we performed in order to directly assess the usefulness of the generated clusters from a practitioner’s perspective. As we explained in Section 4.5.4.3, we surveyed one expert per case study due to the special criteria that potential respondents had to meet, but covered multiple clusters in each survey to mitigate potential expert errors.

**External validity.** We applied our approach to three case studies drawn from two industry sectors. The consistency seen across the results of the case studies provides confidence about the generalizability of our results. Further case studies are nonetheless necessary for improving external validity.

## 4.7 Conclusion

In this chapter, we presented a tool-supported approach for extracting candidate glossary terms from natural language requirements and grouping these terms into clusters based on (syntactic and semantic) similarity between terms. We reported on the application of our approach to three industrial case studies. Our evaluation demonstrated that our approach is significantly more accurate than generic term extraction tools, for glossary terms extraction from requirements documents. Further, the interview surveys conducted with the subject experts in our case studies suggest that our clustering technique offers practical benefits for the construction of requirements glossaries. Based on our evaluation, we devised practical guidelines on tuning the number of clusters variable for a given requirements document. This is important for a successful application of our approach in industry.

In the future, we plan to conduct empirical studies aimed at determining whether our approach leads to compelling quality improvements and cost reductions in comparison with the situation where no automated assistance is used during requirements glossary construction. Finally, we would like to look into additional avenues for utilizing our approach. In particular, we intend to investigate whether our approach can be a useful decision aid for exploring the relationships between the terminologies of requirements documents that originate from different sources.

# Chapter 5

## Extracting Domain Models from Natural-Language Requirements

Domain modeling is an important step in the transition from natural-language requirements to precise and analyzable specifications [Yue et al., 2011]. By capturing in an explicit manner the key concepts of an application domain and the relations between these concepts, a domain model serves both as an effective tool for improving communication between the stakeholders of a proposed application, and further as a basis for detailed requirements and design elaboration [Larman, 2005, Schneider, 2009].

For large systems, building a domain model manually is a laborious task. Several approaches exist to assist engineers with this task, whereby candidate domain model elements are automatically extracted using Natural Language Processing (NLP). Despite the existing work on domain model extraction, important facets remain under-explored: (1) there is limited empirical evidence about the usefulness of existing extraction rules (heuristics) when applied in industrial settings; (2) existing extraction rules do not adequately exploit the natural-language dependencies detected by modern NLP technologies; and (3) an important class of rules developed by the information retrieval community for information extraction remains unutilized for building domain models.

Motivated by addressing the above limitations, we develop a domain model extractor by bringing together existing extraction rules in the software engineering literature, extending these rules with complementary rules from the information retrieval literature, and proposing new rules to better exploit results obtained from modern NLP dependency parsers. We apply our model extractor to four industrial requirements documents, reporting on the frequency of different extraction rules being applied. We conduct an expert study over one of these documents, investigating the accuracy and overall effectiveness of our domain model extractor.

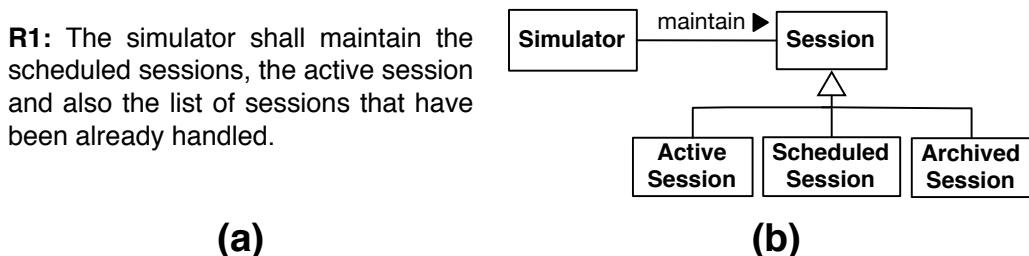
**Structure.** Section 5.1 discusses the motivation and our contributions for the work presented in this chapter. Sections 5.2 and 5.3 review the state of the art and provide background. Section 5.4 presents our domain model extraction approach. Section 5.5 reports on an empirical evaluation of the approach. Section 5.6 presents the summary of the chapter and the directions for future work.

## 5.1 Motivation and Contributions

Natural language (NL) is used prevalently for expressing systems and software requirements [Pohl and Rupp, 2011]. Depending on the development methodology being followed, requirements may be written in different formats, e.g., declarative “shall” statements, use case scenarios, user stories, and feature lists [Pohl and Rupp, 2011]. Certain restrictions, e.g., templates [Arora et al., 2015a, Yue et al., 2015], may be enforced over NL requirements to mitigate ambiguity and vagueness, and to make the requirements more amenable to analysis. In a similar vein, and based on the application context, the engineers may choose among several alternative notations for domain modeling. These notations include, among others, ontology languages such as OWL, entity-relationship (ER) diagrams, UML class diagrams, and SysML block definition diagrams [Ambler, 2004, Holt et al., 2011].

Irrespective of the format in which the requirements are expressed and the notation used for domain modeling, the engineers need to make sure that the requirements and the domain model are properly aligned. To this end, it is beneficial to build the domain model before or in tandem with documenting the requirements. Doing so, however, may not be possible due to time and resource constraints. Particularly, in many industry domains, e.g., aerospace which motivates our work in this chapter, preparing the requirements presents a more immediate priority for the engineers. This is because the requirements are a direct prerequisite for the contractual aspects of development, e.g., tendering and commissioning. Consequently, the engineers may postpone domain modeling to later stages when the requirements have sufficiently stabilized and met the early contractual demands of a project. Another obstacle to building a domain model early on in a complex project is the large number of stakeholders that may be contributing to the requirements, and often the involvement of different companies with different practices.

Building a domain model that is aligned with a given set of requirements necessitates that the engineers examine the requirements and ensure that all the concepts and relationships relevant to the requirements are included in the domain model. This is a laborious task for large applications, where the requirements may constitute tens or hundreds of pages of text. Automated assistance for domain model construction based on NL requirements is therefore important.



**Figure 5.1.** (a) Example requirements statement and (b) corresponding domain model fragment.

This chapter is concerned with developing an automated solution for extracting domain models from *unrestricted* NL requirements, focusing on the situation where one cannot make strong assumptions about either the requirements' syntax and structure, or the process by which the requirements were elicited. We use *UML class diagrams* for representing the extracted models. To illustrate, consider requirements statement R1 in Figure 5.1(a). This requirement originates from the requirements document of a real simulator application in the satellite domain. Upon manually examining R1, a domain expert sketched the domain model fragment shown in Figure 5.1(b). Using heuristic rules

**R2:** The simulator shall connect only to those **networks for** which the **IP addresses** have been **specified**.

(a)



(b)

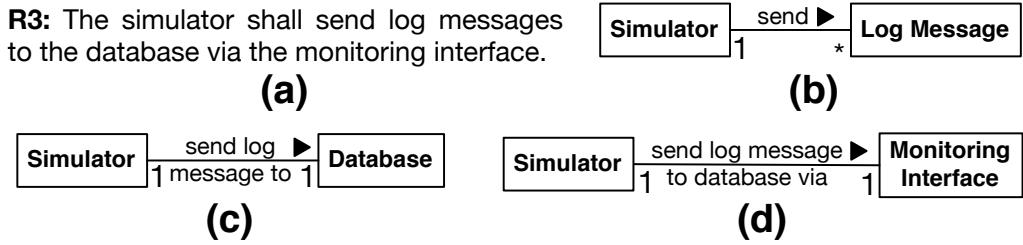
Figure 5.2. (a) Example relative clause modifier (rcmod) dependency and (b) corresponding relation.

implemented via Natural Language Processing (NLP) [Indurkha and Damerau, 2010], a tool could automatically identify several of the elements in this model fragment. Indeed, generalizable rules can be provided to extract all the elements, except for `Archived Session` and its relation to `Session`, whose identification would require human input.

Automated extraction of models from NL requirements has been studied for a long time, with a large body of literature already existing in the area, e.g., [Deeptimahanti and Sanyal, 2011, Elbendak et al., 2011, Ibrahim and Ahmad, 2010, Popescu et al., 2008, Vidya Sagar and Abirami, 2014, Yue et al., 2015], to note some. Nevertheless, important aspects and opportunities that are crucial for the application of model extraction in industry remain under-explored. Notably:

- There is limited empirical evidence about how well existing model extraction approaches perform when applied over industrial requirements. Existing approaches often assume restrictions on the syntax and structure of NL requirements [Yue et al., 2011]. In many industrial situations, e.g., when there are time pressures or little control over the requirements authoring process, these restrictions may not be met [Arora et al., 2015a]. There is therefore a need to empirically study the usefulness of model extraction over unrestricted NL requirements.
- Modern NLP parsers provide detailed information about the dependencies between different segments of sentences. Our examination of existing model extraction rules indicates that there are important dependency types which are detectable via NLP, but which are not currently being exploited for model extraction. To illustrate, consider requirements statement R2, shown in Figure 5.2(a), from the simulator application mentioned earlier. In R2, there is a dependency, called a relative clause modifier (rcmod) dependency [De Marneffe and Manning, 2008], between the phrases “network” and “specified”. Based on this dependency, which is detected by parsers such as the Stanford Parser [Klein and Manning, 2016], one can extract the relation in Figure 5.2(b). Existing model extraction approaches do not utilize rcmod and thus do not find this relation. Similar gaps exist for some other dependency types.
- An important generic class of information extraction rules from the information retrieval literature is yet to be explored for model extraction. This class of rules, referred to as *link paths* [Akbik and Broß, 2009] (or *syntactic constraints* [Fader et al., 2011]), enables extracting relations between concepts that are only *indirectly* related. To illustrate, consider requirements statement R3, shown in Figure 5.3(a), again from the simulator application mentioned earlier. Existing model extraction approaches can detect the relation shown in Figure 5.3(b), as “simulator” and “log message” are directly related to each other by being the subject and the object of the verb “send”, respectively. Nevertheless, existing approaches miss the indirect relations of Figure 5.3(c),(d), which are induced by link paths.

Link path relations can have different *depths*, where the depth represents the number of additional concepts linked to the direct relation. For example, in the relation of Figure 5.3(c), one additional



**Figure 5.3.** (a) Example requirements statement, (b) direct relation, (c-d) link path (indirect) relations.

concept, namely “database”, has been linked to the direct relation, i.e., Figure 5.3(b). The depth of the link path relation is therefore one. Using a similar reasoning, the depth of the link path relation in Figure 5.3(d) is two.

As suggested by our example, the direct relation of Figure 5.3(b) is not the only plausible choice to consider for inclusion in the domain model; the indirect relations of Figure. 5.3(c),(d) present meaningful alternative (or complementary) relations. Indeed, among the three relations in Figure. 5.3(b)-(d), the domain expert found the one in Figure 5.3(c), i.e., the link path of depth one, useful for the domain model and excluded the other two relations. Using link paths in model extraction is therefore an important avenue to explore.

**Contributions.** Motivated by addressing the limitations outlined above, we make the following contributions:

(1) We develop an automated domain model extractor for unrestricted NL requirements. We use UML class diagrams for representing the results of extraction. Our model extractor combines existing extraction rules from the software engineering literature with link paths from the information retrieval literature. We further propose new rules aimed at better exploiting the dependency information obtained from NLP dependency parsers.

(2) Using four industrial requirements documents, we examine the number of times each of the extraction rules implemented by our model extractor is triggered, providing insights about whether and to what extent each rule is capable of deriving structured information from NL requirements in realistic settings. The four documents that we consider collectively contain 786 “shall” requirements statements.

(3) We report on an expert review of the output of our model extractor over 50 randomly-selected requirements

statements from one of the four industrial documents mentioned above. The results of this review suggest that  $\approx 90\%$  of the conceptual relations identified by our model extractor are either correct or partially correct, i.e., having only minor inaccuracies. Such level of correctness, noting that no particular assumptions were made about the syntax and structure of the requirements statements, is promising. At the same time, we observe that, from the set of relations identified, only  $\approx 36\%$  are relevant, i.e., deemed useful for inclusion in the domain model. Our results imply that low relevance is not a shortcoming in our model extractor per se, but rather a broader challenge to which other rule-based model extractors are also prone. In this sense, our expert review reveals an important area for future improvement in automated model extraction.

## 5.2 State of the Art

We synthesize the literature on domain model extraction and compile a set of extraction rules (heuristics) in order to establish the state of the art. The rules identified through our synthesis are shown in Table 5.1. These rules are organized into four categories, based on the nature of the information they extract (concepts, associations and generalizations, cardinalities, and attributes). We illustrate each rule in the table with an example. We note that the literature provides rules for extracting (class) operations as well. However, and in line with best practice [Larman, 2005], we deem operations to be outside the scope of domain models. Furthermore, since operations typically become known only during the design stages, there is usually little information to be found about operations in requirements documents.

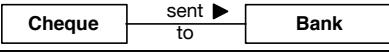
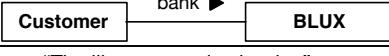
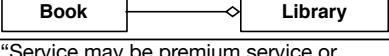
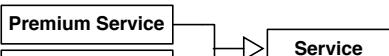
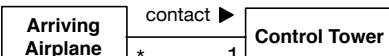
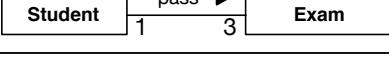
Our focus being on unrestricted NL requirements, we have excluded from Table 5.1 rules that rely on specific sentence patterns. We do nevertheless include in the table five pattern-based rules (B3 to B5 and D1 to D2) due to the generic nature of these rules. The criterion we applied for the inclusion of a pattern-based rule was that the rule must have been considered in at least two distinct previous publications. We further exclude from Table 5.1 rules rooted in programming conventions, e.g., the convention of separating concepts and attributes by an underscore, e.g., *Bank\_Id*.

Next, we describe the sources from which the rules of Table 5.1 originate: The pioneering studies by Abbott [Abbott, 1983] and Chen [Chen, 1983] laid the foundation for the subsequent work on model extraction from textual descriptions. Yue et al. [Yue et al., 2011] survey 20 approaches aimed at transforming textual requirements into early analysis models. Of these, five [Ambriola and Gervasi, 2006, Harmain and Gaizauskas, 2003, Liu et al., 2004, Liu et al., 2003, Mich, 1996] provide automated support for extracting domain models, or models closely related to domain models, e.g., object models. Yue et al. [Yue et al., 2011] bring together the rules from the above approaches, further accounting for the extensions proposed to Abbott's original set of rules [Abbott, 1983] in other related studies. Rules A1 to A4, B1, B4, B5, C1 to C4, and D1 to D3 in Table 5.1 come from Yue et al. [Yue et al., 2011].

To identify more recent strands of related research, we examined all the citations to Yue et al. [Yue et al., 2011] based on Google Scholar. Our objective was to identify any new extraction rules in the recent literature not already covered by Yue et al. [Yue et al., 2011]. We found two publications containing new rules: Vidya Sagar and Abirami [Vidya Sagar and Abirami, 2014], and Ben Abdessalem Karaa et. al. [Ben Abdessalem Karaa et al., 2015]. Our study of Vidya Sagar and Abirami [Vidya Sagar and Abirami, 2014] and a closely-related publication by Elbendak et. al. [Elbendak et al., 2011] upon which Vidya Sagar and Abirami build yielded four new rules. These are A5, B2, B3, and D4 in Table 5.1. As for Ben Abdessalem Karaa et. al. [Ben Abdessalem Karaa et al., 2015], all the new rules proposed therein are pattern-based. These rules do not match our inclusion criterion mentioned above, as no other publication we know of has used these (pattern-based) rules.

A limitation in the rules of Table 5.1 is that these rules do not cover link paths, as we already explained in Section 5.1. Link-path rules have been used in the information retrieval domain for mining structured information from various natural-language sources, e.g., Wikipedia pages [Akbik and Broß, 2009, Fader et al., 2011] and the biomedical literature [Yang et al., 2010]. However, this class of rules has not been used for model extraction to date. Another limitation, again explained in Section 5.1, is that existing model extraction rules do not fully exploit the results from NLP tools.

Table 5.1. Existing domain model extraction rules.

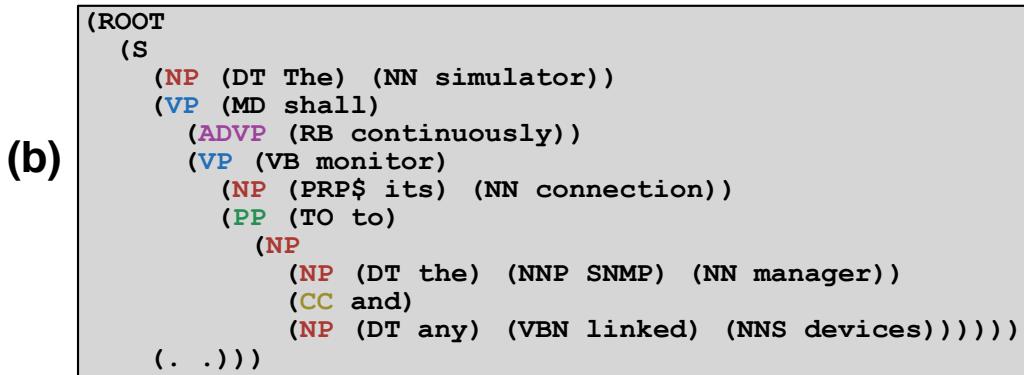
Rule	Description	Example
<b>Concepts</b>	<b>A1</b> All NPs* in the requirements are candidate concepts.	R3 in Figure 3 :: <b>Simulator</b> , <b>Log Message</b> , <b>Database</b> , and <b>Monitoring Interface</b>
	<b>A2</b> Recurring NPs are concepts.	R3 in Figure 3 :: <b>Simulator</b> (if it is recurring)
	<b>A3</b> Subjects in the requirements are concepts.	R3 in Figure 3 :: <b>Simulator</b>
	<b>A4</b> Objects in the requirements are concepts.	R3 in Figure 3 :: <b>Log Message</b>
	<b>A5</b> Gerunds in the requirements are concepts.	"Borrowing is processed by the staff." :: <b>Borrowing</b>
<b>Associations and Generalizations</b>	<b>B1</b> Transitive verbs are associations.	R3 in Figure 3 :: 
	<b>B2</b> A verb with a preposition is an association.	"The cheque is sent to the bank." :: 
	<b>B3</b> <R> in a requirement of the form "<R> of <A> is <B>" is likely to be an association.	"The bank of the customer is BLUX." :: 
	<b>B4</b> "contain", "is made up of", "include", [...] suggest aggregations / compositions.	"The library contains books." :: 
	<b>B5</b> "is a", "type of", "kind of", "may be", [...] suggest generalizations.	"Service may be premium service or normal service." :: 
<b>Cardinalities</b>	<b>C1</b> If the source concept of an association is plural / has a universal quantifier and the target concept has a unique existential quantifier, then the association is many-to-one.	"All arriving airplanes shall contact the control tower." :: 
	<b>C2</b> If the source concept of an association is singular and the target concept is plural / quantified by a definite article, then the association is one-to-many.	"The student passed the exams." :: 
	<b>C3</b> If the source concept of an association is singular and the target concept is singular as well, then the association is one-to-one.	"The student passed the exam." :: 
	<b>C4</b> An explicit number before a concept suggests a cardinality.	"The student passed 3 exams." :: 
<b>Attributes</b>	<b>D1</b> "identified by", "recognized by", "has" [...] suggest attributes.	"An employee is identified by the employee id." :: <b>Employee Id</b> is an attribute of <b>Employee</b> .
	<b>D2</b> Genitive cases, e.g., NP's NP, suggest attributes.	"Book's title" :: <b>Title</b> is an attribute of <b>Book</b> .
	<b>D3</b> The adjective of an adjectively modified NP suggests an attribute.	"large library" :: <b>Size</b> is an attribute of <b>Library</b> .
	<b>D4</b> An intransitive verb with an adverb suggests an attribute.	"The train arrives in the morning at 10 AM." :: <b>Arrival time</b> is an attribute of <b>Train</b> .

\* NP stands for noun phrase; a definition is provided in Section 3.

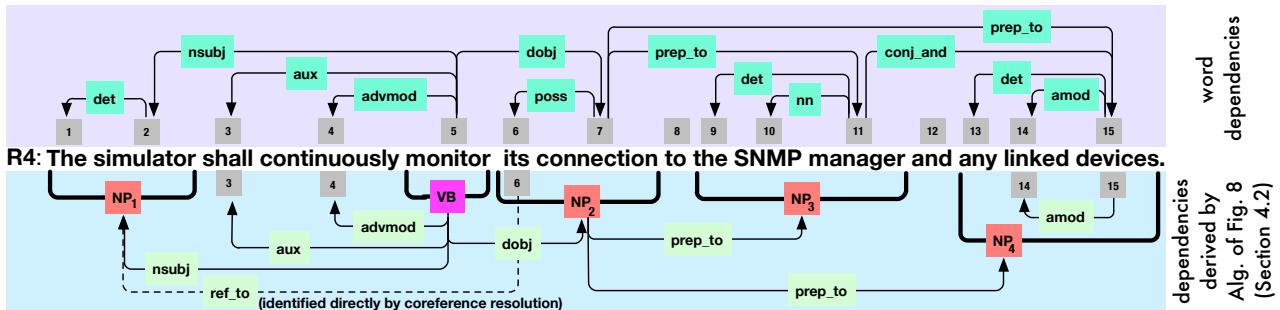
Our approach, described in Section 5.4, proposes extensions in order to address these limitations. Our empirical evaluation, described in Section 5.5, demonstrates that our extensions are of practical significance.

Further, the large majority of existing work on model extraction is evaluated over exemplars and in artificial settings. Empirical studies on model extraction in real settings remain scarce. Our empirical evaluation, which is conducted in an industrial context, takes a step towards addressing this gap.

**(a)** **R4:** The simulator shall continuously monitor its connection to the SNMP manager and any linked devices.



**Figure 5.4.** (a) A requirement and (b) its parse tree.



## 5.3 Syntactic Parsing

In this section, we provide background on *syntactic parsing*, also known as syntactic analysis, which is the key enabling NLP technology for our model extraction approach. Syntactic parsing encompasses two tasks: phrase structure parsing and dependency parsing. Our model extractor uses both. We briefly introduce these tasks below.

*Phrase structure parsing* [Indurkhya and Damerau, 2010] is aimed at inferring the structural units of sentences. We already discussed phrase structure parsing (NLP parsing) in Chapter 3 (see Section 3.3.3). In this chapter, we are interested in the *noun phrases* and *verbs* identified via phrase structure parsing. Recall that a noun phrase (NP) is a unit that can be the subject or the object of a verb. A verb (VB) appears in a verb phrase (VP) alongside any direct or indirect objects, but not

the subject. Verbs can have auxiliaries and modifiers (typically adverbs) associated with them. To illustrate, consider requirements statement R4 in Figure 5.4(a). The structure of R4 is depicted in Figure 5.4(b) as a parse tree. Here, we do not visualize the tree, and instead show it in a nested-list representation commonly used for parse trees.

*Dependency parsing* [Smith, 2011] is aimed at finding grammatical dependencies between the individual words in a sentence. In contrast to phrase structure parsing, which identifies the structural constituents of a sentence, dependency parsing identifies the functional constituents, e.g., the subject and the object. The output of dependency parsing is represented as a directed acyclic graph, with labeled (typed) dependency relations between words. The top part of the graph of Figure 5.5 shows the output of dependency parsing over requirements statement R4. An example typed dependency here is `nsubj(simulator{5}, monitor{2})`, stating that “simulator” is the subject of the verb “monitor”.

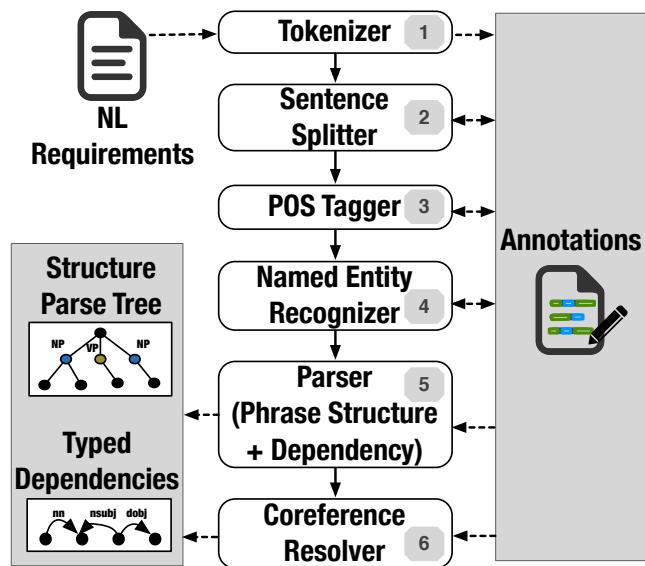


Figure 5.6. Parsing pipeline.

Syntactic parsing is commonly done using the pipeline of NLP modules shown in Figure 5.6. We note that the first four modules in this pipeline are the same as those in the text chunking pipeline discussed in Chapter 3. To maintain the flow, we describe, in the explanation that follows, the pipeline in its entirety, including the first four modules. For the work reported in this chapter, we use the pipeline implementation provided by the Stanford Parser [Klein and Manning, 2016].

The first module in the pipeline is the Tokenizer, which splits the input text, in our context a requirements document, into tokens. A token can be a word, a number, or a symbol. The second module, the Sentence Splitter, breaks the text into sentences. The third module, the POS Tagger, attaches a part-of-speech (POS) tag to each token. POS tags represent the syntactic categories of tokens, e.g., nouns, adjectives and verbs. The fourth module, the Named-Entity Recognizer, identifies entities belonging to pre-defined categories, e.g., proper nouns, dates and locations. The fifth and main module is the Parser, encompassing both phrase structure parsing and dependency parsing. The final module is the Coreference Resolver. This (optional) module finds expressions that refer to the same entity. We concern ourselves with *pronominal* coreference resolution only, which is the task of identifying, for a given pronoun such as “its” and “their”, the NP that the pronoun refers to.

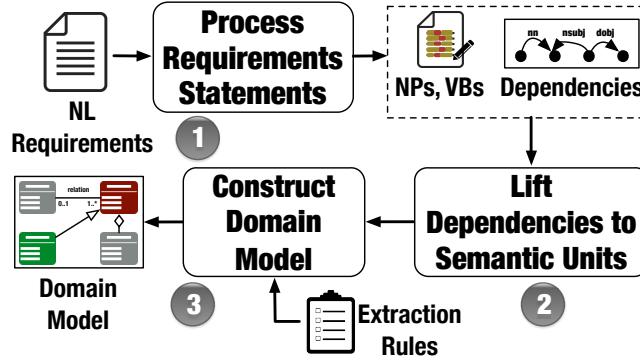


Figure 5.7. Approach Overview.

Figure 5.5 shows an example of pronominal coreference resolution, where the pronoun “its” is linked to the referenced NP, namely “the simulator”, via a `ref_to` dependency.

## 5.4 Approach

Figure 5.7 presents an overview of our domain model extraction approach. The input to the approach is an NL requirements document and the output is a UML class diagram. Below, we elaborate the three main steps of our approach, marked 1-3 in Figure 5.7.

### 5.4.1 Processing the Requirements Statements

The requirements processing step includes the following activities: (1) detecting the phrasal structure of the requirements, (2) identifying the dependencies between the words in the requirements, (3) resolving pronominal coreferences, and (4) performing stopword removal and lemmatization; these are common NLP tasks, respectively for pruning words that are unlikely to contribute to text analysis, and for transforming words into their base morphological form.

Activities (1), (2), and (3) are carried out by the pipeline of Figure 5.6. From the parse tree generated by this pipeline for each requirements statement, we extract the *atomic* NPs and the VBs. Atomic NPs are those that cannot be further decomposed. For example, from the parse tree of Figure 5.4(b), we extract: “The simulator” (NP), “monitor” (VB), “its connection” (NP), “the SNMP manager” (NP) and “any linked devices” (NP). We do not extract “the SNMP manager and any linked devices” because this NP is not atomic.

We then subject the NPs to stopword removal. Stopwords are words that appear so frequently in the text that they no longer serve an analytical purpose [Manning and Schütze, 1999]. Stopwords include, among other things, determiners and predeterminers. In our example, stopword removal strips the extracted NPs of the determiners “the” and “any”. The VBs and the tail words of the NPs are further subject to lemmatization. In our example, “linked devices” is transformed into “linked device”. Had the VB been, say, “monitoring”, it would have been transformed into “monitor”. VBs in passive form are an exception and not lemmatized, e.g., see the example of Figure 5.2.

The NPs and VBs obtained by following the process above provide the initial basis for labeling the concepts, attributes, and associations of the domain model that will be constructed in Step 3 of our approach (see Figure 5.7). The dependencies obtained from executing the pipeline of Figure 5.6

need to undergo further processing and be combined with the results of coreference resolution before they can be used for domain model construction. This additional processing is addressed by Step 2 of our approach, as we explain next in Section 5.4.2.

### 5.4.2 Deriving Dependencies at a Semantic Level

As seen from Figure 5.5, the results of dependency parsing (top of the figure) are at the level of words. Many of these dependencies are meaningful for model extraction only at the level of NPs, which, along with verbs, are the main semantic (meaning-bearing) units of sentences. For example, consider the dependency `prep_to(connection{7}, manager{11})`, stating that “manager” is a prepositional complement to “connection”. To derive from this dependency a meaningful relation for the domain model, we need to raise the dependency to the level of the involved NPs, i.e., `prep_to(NP2, NP3)` in Figure 5.5. We do so using the algorithm of Figure 5.8.

The algorithm takes as input a set  $P$  composed of the atomic NPs and the VBs, and the results of dependency parsing and coreference resolution, all from Step 1 of our approach (Section 5.4.1). The algorithm initializes the output (i.e., the semantic-unit dependencies) with the `ref_to` dependencies (L.1), noting that the targets of `ref_to` dependencies are already at the level of NPs. Next, the algorithm identifies, for each word dependency, the element(s) in  $P$  to which the source and the target of the dependency belong (L.3–12). If either the source or target words fall outside the boundaries of the elements in  $P$ , the words themselves are treated as being members of  $P$  (L.6,11). This behavior serves two purposes: (1) to link the VBs to their adverbial modifiers, illustrated in the example of Figure 5.5, and (2) to partially compensate for mistakes made by phrase structure parsers, which are typically only  $\approx 90\%$  accurate in phrase detection [Attardi and Dell’Orletta, 2008, Zhu et al., 2013]. Dependencies between the constituent words of the same NP are ignored (L.13), except for the adjectival modifier (`amod`) dependency (L.16–18) which is used by rule D3 of Table 5.1. When the algorithm of Figure 5.8 is executed over our example requirements statement R4, it yields the dependencies shown on the bottom of Figure 5.5. These dependencies are used in Step 3 of our approach, described next, for extracting associations, aggregations, generalizations and also for linking attributes to concepts.

### 5.4.3 Domain Model Construction

The third and final step of our approach is constructing a domain model. This step uses the NPs and VBs identified in Step 1 (after stopword removal and lemmatization), along with the semantic-unit dependencies derived in Step 2. The extraction rules we apply for model construction are: (1) the rules of Table 5.1 gleaned from the state of the art, (2) three new rules, described and exemplified in Table 5.2, which we propose in order to exploit dependency types that have not been used for model extraction before, and (3) link paths [Akbik and Broß, 2009], which we elaborate further later in this section. In Table 5.3, we show all the model elements that our approach extracts from our example requirements statement R4. In the rest of this section, we outline the main technical factors in our domain model construction process. We organize our discussion under five headings: domain concepts, associations, generalizations, cardinalities, and attributes.

**Domain concepts.** All the extracted NPs (from Step 1) are initially considered as candidate concepts. If a candidate concept appears as either the source or the target of some dependency (from Step 2), the candidate concept will be marked as a domain concept. If either the source or the target

**Input:** A set  $P$  of all (atomic) NPs and VBs in a sentence  $S$ ;

**Input:** A set  $D_{Word}$  of word dependencies in  $S$ ;

**Input:** A set  $R$  of `ref_to` dependencies for the pronouns in  $S$ ;

**Output:** A set  $D_{Sem}$  of semantic-unit dependencies for  $S$ ;

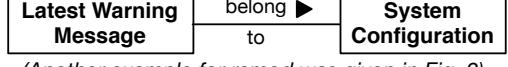
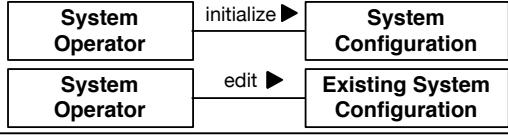
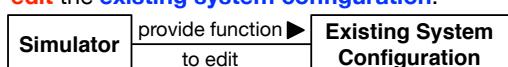
```

1:  $D_{Sem} \leftarrow R$ ; /*Initialize  $D_{Sem}$  with the results of coref resolution.*/
2: for all  $dep \in D_{Word}$  do
3:   if (there exists some  $p \in P$  to which  $dep.source$  belongs) then
4:      $p_{source} \leftarrow p$ ;
5:   else
6:      $p_{source} \leftarrow dep.source$ ;
7:   end if
8:   if (there exists some  $p \in P$  to which  $dep.target$  belongs) then
9:      $p_{target} \leftarrow p$ ;
10:  else
11:     $p_{target} \leftarrow dep.target$ ;
12:  end if
13:  if ( $p_{source} \neq p_{target}$ ) then
14:    Add to  $D_{Sem}$  a new dependency with source  $p_{source}$ ,
        target  $p_{target}$  and type  $dep.type$ ;
15:  else
16:    if ( $dep.type$  is amod) then
17:      Add  $dep$  to  $D_{Sem}$ ;
18:    end if
19:  end if
20: end for
21: return  $D_{Sem}$ 

```

**Figure 5.8.** Algorithm for lifting word dependencies to semantic-unit dependencies.

**Table 5.2.** New extraction rules in our approach.

Rule	Description	Example
<b>N1</b>	Relative clause modifiers of nouns ( <code>rcmod</code> dependency) suggest associations.	<p>“The system operator shall display the <b>system configuration</b>, to which the <b>latest warning message belongs</b>.”</p>  <p>(Another example for <code>rcmod</code> was given in Fig. 2)</p>
<b>N2</b>	Verbal clausal complements ( <code>ccomp/xcomp</code> dependencies) suggest associations.	<p>“The <b>system operator</b> shall be able to <b>initialize</b> the <b>system configuration</b>, and to <b>edit</b> the <b>existing system configuration</b>.”</p> 
<b>N3</b>	Non-finite verbal modifiers ( <code>vmod</code> dependencies) suggest associations.	<p>“The <b>simulator</b> shall <b>provide a function to edit</b> the <b>existing system configuration</b>.”</p> 

end of a dependency is a pronoun, that end is treated as being the concept to which the pronoun refers. Table 5.1 lists a total of five rules, A1–A5, for identifying domain concepts. Our approach implements A1, which subsumes A2–A5. We enhance A1 with the following procedure for NPs that

Table 5.3. Extraction results for R4 of Figure 5.4(a).

#	Type of element(s) extracted	Extracted element(s)	Extraction rule(s) triggered
1	Candidate Concept	Simulator, Connection, SNMP Manager, Linked Device, Device	A1*
2	Association	Simulator   continuously monitor   Connection	B1, C3 (for cardinalities)
3	Association	Simulator   continuously monitor   SNMP Manager	Link Path Depth 1, C3 (for cardinalities)
4	Association	Simulator   continuously monitor   Linked Device	Link Path Depth 1, C2 (for cardinalities)
5	Aggregation	Connection  ---◇  Simulator	D2† (+ coreference resolution)
6	Generalization	Linked Device  ---▷  Device	D3†

\*A1 in this table is an enhanced version of A1 in Table 1, as discussed in Section 4.3.

†As we explain in Section 4.3, in contrast to some existing approaches, we use D2 and D3 (from Table 1) for extracting aggregations and generalizations, respectively, rather than attributes.

Table 5.4. Different subject types.

Subject Type	Example	Subject
Simple Subject	"The operator shall initialize the system configuration."	operator
Passive Subject	"The system configuration shall be initialized by the operator."	operator
Genitive Subject	"The operator of the ground station shall initialize the system configuration."	operator

have an adjectival modifier (amod dependency), as long as the adjective appears at the beginning of an NP after stopword removal: we remove the adjective from the NP and add the remaining segment of the NP as a domain concept. For example, consider row 1 of Table 5.3. The concept of Device here was derived from Linked Device by removing the adjectival modifier. The relation between Device and Linked Device is established via rule D3 discussed later (see *Generalizations*).

**Associations.** The VBs (from Step 1) that have subjects or objects or both give rise to associations. The manifestation of the subject part may vary in different sentences. Table 5.4 lists and exemplifies the subject types that we handle in our approach. Our treatment unifies and generalizes rules B1 and B2 of Table 5.1. We further implement rule B3, but as we observe in our evaluation (Section 5.5), this rule is not useful for the requirements in our case studies.

For extracting associations, we further propose three new rules, N1–N3, shown in Table 5.2. Rule N1 utilizes relative clause modifier (rcmod) dependencies. In the example provided for this rule in Table 5.2, “system configuration” is modified by a relative clause, “to which the latest warning message belongs”. From this, N1 extracts an association between System Configuration and Latest Warning Message.

Rule N2 utilizes verbal clausal complement (ccomp and xcomp) dependencies. In the example given in Table 5.2, “initialize” and “edit” are clausal complements to “able”. Here, the subject, “system operator”, is linked to “able”, and the two objects, “system configuration” and “existing system configuration”, are linked to “initialize” and “edit”, respectively. What N2 does here is to infer that “system operator” (conceptually) serves as a subject to “initialize” and “edit”, extracting the two associations shown in Table 5.2.

As for Rule N3, the purpose is to utilize non-finite verbal modifier (vmod) dependencies. In the example we show in Table 5.2, “edit” is a verbal modifier of “function”. We use this information for enhancing the direct subject-object relation between “simulator” and “function”. Specifically, we link the object of the verbal modifier, “existing system configuration”, to the subject, “simulator”, extracting an association between Simulator and Existing System Configuration.

The associations resulting from our generalization of B1 and B2, explained earlier, and from our new rules, N1 to N3, are all subject to a secondary process, aimed at identifying link paths [Akbik and Broß, 2009]. Intuitively, a link path is a combination of two or more direct links. To illustrate, consider rows 3 and 4 in Table 5.3. The basis for both of the associations shown in these rows is the direct association in row 2 of the table. The direct association comes from the subject-object relation between  $NP_1$  and  $NP_2$  in the dependency graph of Figure 5.5. The association in row 3 of Table 5.3 is induced by combining this direct association with the dependency  $prep\_to(NP_2, NP_3)$  from the dependency graph. The association of row 4 is the result of combining the direct association with another dependency,  $prep\_to(NP_2, NP_4)$ . In our approach, we consider all possible ways in which a direct association can be combined with paths of prepositional dependencies ( $prep\_*$  dependencies in the dependency graph).

For extracting aggregations, which are special associations denoting containment relationships, we use rules B4 and D2 from Table 5.1. With regard to D2, we point out that a number of previous approaches, e.g., [Elbendak et al., 2011, Yue et al., 2011, Vidya Sagar and Abirami, 2014], have used this rule for identifying attributes. Larman [Larman, 2005] notes the difficulty in choosing between aggregations and attributes, recommending that any entity that represents in the real world something other than a number or a string of text should be treated as a domain concept, rather than an attribute. Following this recommendation, we elect to use D2 for extracting aggregations. Ultimately, the user will need to decide which representation, an aggregation or an attribute, is most suitable on a case-by-case basis.

An important remark about D2 is that this rule can be combined with coreference resolution, which to our knowledge, has not been done before for model extraction. An example of this combination is given in row 5 of Table 5.3. Specifically, the aggregation in this row is induced by the *possessive* pronoun “its”, which is linked to “simulator” via coreference resolution (see the  $ref\_to$  dependency in Figure 5.5).

**Generalization.** From our experience, we surmise that generalizations are typically left tacit in NL requirements and are thus hard to identify automatically. The main rule targeted at generalizations is B5 in Table 5.1. This rule, as evidenced by our evaluation (Section 5.5), has limited usefulness when no conscious attempt has been made by the requirements authors to use the patterns in the rule.

We nevertheless observe that certain generalizations manifest through adjectival modifiers. These generalizations can be detected by rule D3 in Table 5.1. For example, row 6 of Table 5.3 is extracted by D3. Like rule D2 discussed earlier, D3 has been used previously for attributes. However, without user intervention, identifying attributes using D3 poses a challenge. To illustrate, consider the example we gave in Table 5.1 for D3. There, the user would need to provide the attribute name, `size`. For simple cases, e.g., sizes, colors, shapes and quantities, one can come up with a taxonomy of adjective types and use the type names as attribute names [Vidya Sagar and Abirami, 2014]. We observed though that generic adjective types are unlikely to be helpful for real and complex requirements. We therefore elect to use D3 for extracting generalizations. Similar to the argument we gave for D2, we leave it to the user to decide when an attribute is more suitable and to provide an attribute name when this is the case.

**Cardinalities.** We use rules C1 to C4 of Table 5.1 for determining the cardinalities of associations. These rules are based on the quantifiers appearing alongside the terms that represent domain concepts, and the singular versus plural usage of these terms. For example, the cardinalities shown in rows 2 to 4 of Table 5.3 were determined using these rules.

**Attributes.** We use rules D1 and D4 of Table 5.1 for extracting attributes. As we discussed above, we have chosen to use rules D2 and D3 of Table 5.1 for extracting aggregations and generalizations, respectively. With regard to rule D4, we note that one cannot exactly pinpoint the name of the attribute using this rule. Nevertheless, unlike rule D3 which is not applicable without user intervention or an adjective taxonomy, D4 can reasonably guess the attribute name. For instance, if we apply our implementation of D4 to the requirement exemplifying this rule in Table 5.1, we obtain `arrive` (instead of `arrival time`) as the attribute name.

## 5.5 Empirical Evaluation

In this section, we evaluate our approach by addressing the following Research Questions (RQs):

**RQ1. How frequently are different extraction rules triggered?** One cannot expect large gains from rules that are triggered only rarely. A rule being triggered frequently is thus an important prerequisite for the rule being useful. RQ1 aims to measure the number of times different extraction rules are triggered over industrial requirements.

**RQ2. How useful is our approach?** The usefulness of our approach ultimately depends on whether practitioners find the approach helpful in real settings. RQ2 aims to assess through a user study the correctness and relevance of the results produced by our approach.

**RQ3. Does our approach run in practical time?** Requirements documents may be large. One should thus be able to perform model extraction quickly. RQ3 aims to study whether our approach has a practical running time.

**Table 5.5.** Description of case study documents.

Case	Description	# of reqs.	Template used?	% of reqs. complying with template
Case A	Simulator application for satellite systems	158	No	N.A.
Case B	Satellite ground station control system	380	Yes (Rupp's)	64%
Case C	Satellite data dissemination network	138	No	N.A.
Case D	Safety evidence information management system for safety certification	110	Yes (Rupp's)	89%

**Table 5.6.** Number of times extraction rules were triggered and number of extracted elements (per document).

	A1*	B1	B2	B3	B4	B5	C1-4	D1	D2	D3	D4	N1	N2	N3	LP	# of concepts	# of attributes <sup>†</sup>	# of (regular) associations	# of aggregations	# of generalizations
Case A	370	139	20	0	24	1	526	0	47	76	4	31	48	42	246	370	4	526	71	77
Case B	620	210	19	0	9	1	730	1	81	89	25	58	69	47	327	620	26	730	90	90
Case C	541	68	17	0	5	2	405	0	85	130	21	77	36	6	201	541	21	405	90	132
Case D	85	40	7	0	2	0	274	0	41	35	15	23	11	68	125	85	15	274	43	35

\*A1 in this table is an enhanced version of A1 in Table 1, as discussed in Section 4.3. A1 subsumes A2 to A5 (of Table 1), as noted in the same section.

†The small number of attributes is explained by our decision to use D2 and D3 (resp.) for extracting aggregations and generalizations instead of attributes, as noted in Section 4.3.

## 5.5.1 Implementation

For syntactic parsing and coreference resolution, we use Stanford Parser [Klein and Manning, 2016]. For lemmatization and stopword removal, we use existing modules in the GATE NLP Workbench [GATE, 2016]. We implemented the model extraction rules using GATE’s scripting language, JAPE, and GATE’s embedded Java environment. The extracted class diagrams are represented using logical predicates (Prolog-style facts). Our implementation is approximately 3,500 lines of code, excluding comments and third-party libraries. Our implementation is available at: <https://bitbucket.org/carora03/redomex>.

## 5.5.2 Results and Discussion

Our evaluation is based on four industrial requirements documents, all of which are collections of “shall” requirements written in English. Table 5.5 briefly describes these documents, denoted Case A–D, and summarizes their main characteristics. We use all four documents for RQ1 and RQ3. For RQ2, we use selected requirements from Case A.

Cases A–C concern software systems in the satellite domain. These three documents were written by different teams in different projects. In Cases B and D, the requirements authors had made an effort to comply with Rupp’s template [Pohl and Rupp, 2011], which organizes the structure of requirements sentences into certain pre-defined slots. The number of template-compliant requirements in these two documents is presented in Table 5.5 as a percentage of the total number of requirements. These percentages were computed in our previous work [Arora et al., 2015a], where Cases B and D were also used as case studies. Our motivation to use template requirements in our evaluation of model extraction is to investigate whether restricting the structure of requirements would impact the applicability of generic extraction rules, which assume no particular structure for the requirements.

**RQ1.** Table 5.6 presents the results of executing our model extractor on Cases A–D. Specifically, the table shows the number of times each of the rules employed in our approach has been triggered over our case study documents. The rule IDs correspond to those in Tables 5.1 and 5.2; LP denotes link

**Table 5.7.** Correctness and relevance results obtained from our expert interview, organized by extraction rules.

(Relation) Extraction Rule																			
B1		B2		B4		D2		D3		D4		N1		N2		N3		Link Paths	
Q1 (Correctness)				Q2 (Relevance)				Q1 (Correctness)				Q2 (Relevance)				Q1 (Correctness)			
Y	P	N	C%	Y	P	N	C%	Y	P	N	C%	Y	P	N	C%	Y	P	N	C%
18	11	0	100%	3	1	0	100%	13	0	4	77%	16	4	2	91%	17	0	6	74%
12	0	17	41%	1	0	3	25%	0	0	17	0%	8	0	14	36%	7	4	12	48%
3	1	0	100%	13	0	4	77%	16	4	2	91%	17	0	6	74%	2	4	1	86%
10	10	0	100%	5	9	1	93%	42	26	6	92%	7	0	13	35%	7	1	7	53%
26	0	48	35%	26	0	48	35%	26	0	48	35%	26	0	48	35%	26	0	48	35%

Legend Q1 (Correctness): Y Yes P Partially N No C% Correctness (%) Q2 (Relevance): Y Yes M Maybe N No R% Relevance (%)

paths. The table further shows the number of extracted elements for each case study, organized by element types.

As indicated by Table 5.6, rules B1, C1 to C4, D2, D3, and LP are the most frequently triggered. B1 is a generic rule that applies to all transitive verbs. D2 and D3 address genitive cases and the use of adjectives in noun phrases. These constructs are common in English, thus explaining the frequent application of D2 and D3. We note that, as we explained in Section 5.4.3, we use D2 and D3 for identifying aggregations and generalizations, respectively.

Link paths, as stated earlier, identify indirect associations. Specifically, link paths build upon the direct associations identified by B1, B2, and N1 to N3. To illustrate, consider the example in Figure 5.3. Here, B1 retrieves the direct association in Figure 5.3(b), and link paths retrieve the indirect ones in Figure. 5.3(c),(d). In this example, we count B1 as being triggered once and link paths as being triggered twice.

Rules C1 to C4 apply to all associations, except aggregations. More precisely, C1 to C4 are considered only alongside B1 to B3, N1 to N3, and link paths. For instance, C2 is triggered once and C3 twice for the associations of Figure. 5.3(b-d).

Our results in Table 5.6 indicate that B3, B5, and D1 were triggered either rarely or not at all in our case study documents. These rules are based on fixed textual patterns. While the patterns underlying these rules seem intuitive, our results suggest that, unless the requirements authors have been trained a priori to use the patterns (not the case for the documents in our evaluation), such patterns are unlikely to contribute significantly to model extraction.

With regard to link paths and our proposed rules, N1 to N3, in Table 5.2, we make the following observations: Link paths extracted 47% of the (non-aggregation) associations in Case A, 45% in Case B, 50% in Case C, and 46% in Case D. And, rules N1 to N3 extracted 23% of the associations in Case A, 24% in Case B, 29% in Case C, and 37% in Case D. *These percentages indicate that link paths and our new rules contribute significantly to model extraction.* Assessing the quality of the extracted results is the subject of RQ2.

With regard to whether the use of templates has an impact on the applicability of the generic rules considered in this chapter, *the results in Table 5.6 do not suggest an advantage or a disadvantage for templates, as far as the frequency of the application of the extraction rules is concerned.* We therefore anticipate that the generic rules considered in this chapter should remain useful for restricted requirements too. Placing restrictions on requirements may nevertheless provide opportunities for developing additional extraction rules [Yue et al., 2015]. Such rules would naturally be tied to the specific restrictions enforced and are thus outside the scope of this chapter.

<b>Q1 (asked per relation).</b> Is this relation correct?
<input type="checkbox"/> Yes <input type="checkbox"/> Partially <input type="checkbox"/> No
<b>Q2 (asked per relation).</b> Should this relation be in the domain model?
<input type="checkbox"/> Yes <input type="checkbox"/> Maybe <input type="checkbox"/> No
<b>Q3 (asked per requirements statement).</b> Are there any other relations that this requirements statement implies? If yes, please elaborate.

Figure 5.9. Interview survey questionnaire.

**RQ2.** RQ2 aims at assessing practitioners' perceptions about the correctness and relevance of the results produced by our approach. Our basis for answering RQ2 is an interview survey conducted with the lead requirements analyst in Case A. Specifically, we selected at random 50 requirements statements (out of a total of 158) in Case A and solicited the expert's feedback about the model elements that were automatically extracted from the selected requirements. Choosing Case A was dictated by the criteria we had to meet: To conduct the interview, we needed expert(s) who had UML domain modeling experience and who were further fully familiar with the requirements. This restricted our choice to Cases A and B. Our interview would further require a significant time commitment from the expert(s). Making such a commitment was justified by the expert for Case A only, due to the project still being ongoing.

We collected the expert's feedback using the questionnaire shown in Figure 5.9. This questionnaire has three questions, Q1 to Q3, all oriented around the notion of "relation". We define relations to include (*regular*) *associations*, *aggregations*, *generalizations*, and *attributes*. The rationale for treating attributes as relations is the conceptual link that exists between an attribute and the concept to which the attribute belongs. The notion of relation was clearly conveyed to the expert using a series of examples prior to the interview. Our questionnaire does not include questions dedicated to domain concepts, since, as we explain below, the correctness and relevance of the domain concepts at either end of a given relation are considered while that relation is being examined. During the interview, the expert was asked to evaluate, through Q1 and Q2 in the questionnaire, the individual relations extracted from a given requirements statement. The expert was further asked to verbalize his rationale for the responses he gave to these questions. Once all the relations extracted from a requirements statement had been examined, the expert was asked, through Q3, whether there were any other relations implied by that requirements statement which were missing from the extracted results.

The relations extracted from each requirements statement were presented to the expert in the same visual format as depicted by the third column of Table 5.3. The extraction rules involved were not shown to the expert. To avoid decontextualizing the relations, we did not present to the expert the extracted relations in isolation. Instead, a given requirements statement and all the relations extracted from it were visible to the expert on a single sheet as we traversed the relations one by one and asking Q1 and Q2 for each of them.

Q1 addresses correctness. A relation is deemed correct if the expert can infer the relation by reading the underlying requirements statement. We instructed the expert to respond to Q1 by "Yes" for a given relation, if all the following criteria were met: (1) the concept (or attribute) at each end of the relation is correct, (2) the type assigned to the extracted relation (e.g., association or generalization) is correct, and (3) if the relation represents an association, the label and the cardinalities of

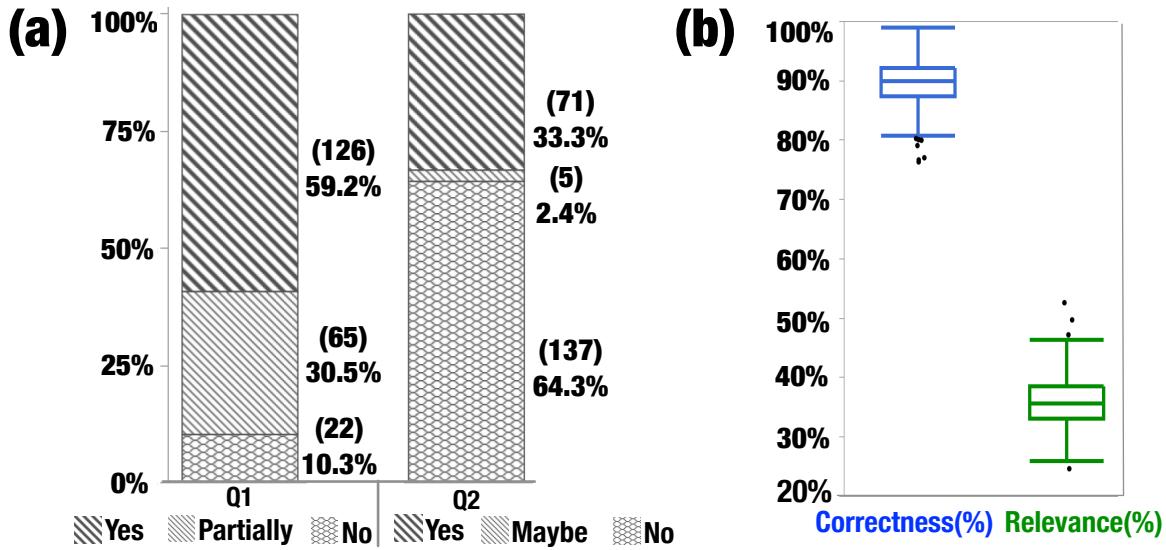


Figure 5.10. (a) Raw and (b) bootstrapping results for Q1 and Q2.

the association are correct. The expert was instructed to answer by “Partially” when he saw some inaccuracy with respect to the correctness criteria above, but he found the inaccuracy to be minor and not compromising the meaningfulness of the relation; otherwise, the expert was asked to respond by “No”.

The correctness of a relation per se does not automatically warrant its inclusion in the domain model. Among other reasons, the relation might be *too obvious* or *too detailed* for the domain model. Q2 addresses relevance, i.e., whether a relation is appropriate for inclusion in the domain model. The expert was asked Q2 for a given relation only upon a “Yes” or “Partially” response to Q1. If the expert’s answer to Q1 was “No”, the answer to Q2 was an automatic “No”. If the expert had answered Q1 by “Partially”, we asked him to answer Q2 assuming that the inaccuracy in the relation had been already resolved.

Finally, Q3 addresses missing relations. A relation is missing if it is identifiable by a domain expert upon manually examining a given requirements statement  $R$ , but which is absent from the relations that have been automatically extracted from  $R$ . A missing relation indicates one or a combination of the following situations: (1) information that is not extracted due to technical limitations in automation, (2) information that is tacit in a requirements statement and thus inferable only by a human expert, (3) information that is implied by the extracted relations, but which the expert decides to represent differently, i.e., using modeling constructs different than the extracted relations. The expert answered Q3 after having reviewed all the relations extracted from a given requirements statement.

Our interview was split into three sessions, with a period of at least one week in between the sessions. The duration of each session was limited to a maximum of two hours to avoid fatigue effects. At the beginning of each session, we explained and exemplified to the expert the interview procedure, including the questionnaire.

Our approach extracted a total of 213 relations from the 50 randomly-selected requirements of Case A. All these 213 relations were examined by the expert. Figure 5.10(a) shows the interview

results for Q1 and Q2. As shown by the figure: First,  $\approx 90\%$  of the relations were deemed correct or partially correct, and the remaining  $10\%$  incorrect; and second,  $\approx 36\%$  of the relations were deemed relevant or maybe relevant for inclusion in the domain model. The remaining  $64\%$  of the relations were deemed not relevant (inclusive of the  $10\%$  of the relations that were deemed incorrect).

Due to the expert's limited availability, we covered only  $\approx 32\%$  (50/158) of the requirements statements in Case A. The 213 relations extracted from these requirements constitute  $\approx 31\%$  (213/678) of the total number of relations obtained from Case A by our model extractor. To provide a measure of correctness and relevance which further accounts for the uncertainty that results from our random sampling of the requirements statements, we provide confidence intervals for correctness and relevance using a statistical technique, known as *bootstrapping* [Efron and Tibshirani, 1994]. Specifically, we built 1000 resamples with replacement of the 50 requirements that were examined in our interview. We then computed as a percentage the correctness and relevance of the relations extracted from each resample. For a given resample, the correctness percentage is the ratio of correct and partially correct relations over the total number of relations. The relevance percentage is the ratio of relevant and maybe relevant relations over the total number of relations. Figure 5.10(b) shows, using box plots, the distributions of the correctness and relevance percentages for the 1000 resamples. *These results yield a 95% confidence interval of 83%–96% for correctness and a 95% confidence interval of 29%–43% for relevance.*

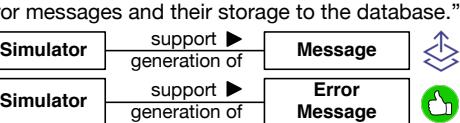
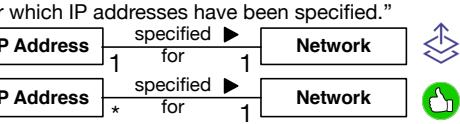
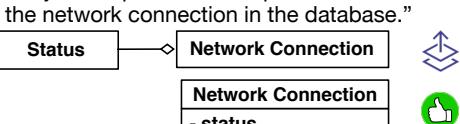
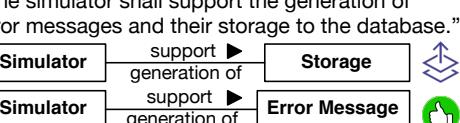
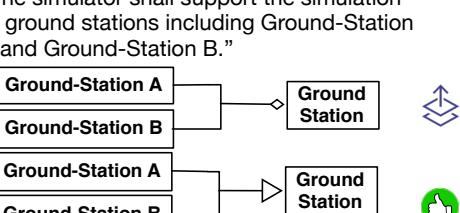
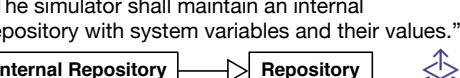
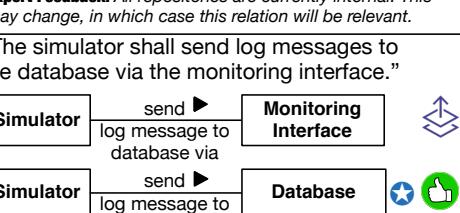
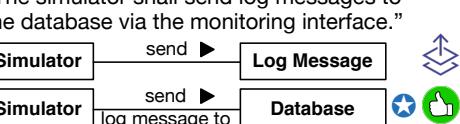
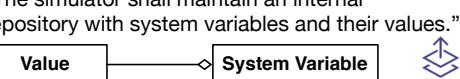
The practical implication of the above findings is that, when reviewing the extracted relations, analysts will have to filter 57%–71% of the relations, despite the large majority of them being correct or partially correct. While we anticipate that filtering the unwanted relations would be more cost-effective than forgoing automation and manually extracting the desired relations from scratch, the required level of filtering needs to be reduced. As we discuss in Section 5.6, improving relevance and minimizing such filtering is an important direction for future work.

In Table 5.7, we provide a breakdown of our interview survey results, organized according to the rules that were triggered over our requirements sample and showing the correctness and relevance percentages for each rule. As seen from these percentages, all triggered rules except B4 and D4 proved useful in our study. In particular, the results of Table 5.7 indicate that our proposed extensions, i.e., rules N1 to N3 and link paths, are useful in practice.

An observation emerging from the relevance percentages in Table 5.7 (green-shaded cells) is that relevance is low across all the extraction rules and not only for our proposed extensions (N1 to N3 and link paths). *This implies that other existing rule-based approaches for domain model extraction are also susceptible to the relevance challenge.* This observation further underscores the need for addressing the relevance challenge in future work.

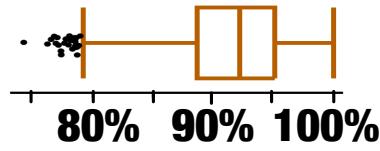
As noted earlier, we asked the expert to verbalize his rationale for his responses to Q1 and Q2. This rationale contained a wealth of information as to what made a relation incorrect or only partially correct, and what made a relation not relevant. In Table 5.8, we provide a classification of the reasons the expert gave for partial correctness and for incorrectness (Q1) and for non-relevance (Q2). The number of relations falling under each category in the classification is provided in the column labeled “Count”. For each category, we provide an example of a problematic relation and, where applicable, the relation desired by the expert. The table is self-explanatory. The only remark to be made is that the reasons given by the expert for partial correctness and for incorrectness have one area of overlap,

Table 5.8. Reasons for inaccuracies and non-relevance.

			#	Reason	Count	Example
Q1	Partially Correct	Incorrect	1	Imprecise label for relation or concept	32	<p>“The simulator shall support the generation of error messages and their storage to the database.”</p> 
			2	Wrong cardinality	29	<p>“The simulator shall connect only to those networks for which IP addresses have been specified.”</p> 
			3	Wrong relation type	4	<p>“The system operator shall update the status of the network connection in the database.”</p> 
	Incorrect	Incorrect	4	Non-existent relation detected	18	<p>“The simulator shall support the generation of error messages and their storage to the database.”</p> 
			5	Wrong relation type	4	<p>“The simulator shall support the simulation of ground stations including Ground-Station A and Ground-Station B.”</p> 
Q2	Maybe Relevant	Incorrect	6	Future contingency	5	<p>“The simulator shall maintain an internal repository with system variables and their values.”</p>  <p><b>Expert Feedback:</b> All repositories are currently internal. This may change, in which case this relation will be relevant.</p>
			7	Relation too detailed	88	<p>“The simulator shall send log messages to the database via the monitoring interface.”</p> 
	Not Relevant	Not Relevant	8	Incomplete constraint	21	<p>“The simulator shall send log messages to the database via the monitoring interface.”</p> 
			9	Obvious / Common knowledge	6	<p>“The simulator shall maintain an internal repository with system variables and their values.”</p> 

★ The “desired” relation is indeed also extracted by our model extractor via a different rule. The goal here is to illustrate what the expert considered to be not relevant.

Legend  Extracted  Desired



**Figure 5.11.** % of relevant relations retrieved.

namely *wrong relation type*, as seen from rows 3 and 5 of Table 5.8. For instance, in the example of row 3, an aggregation was extracted, but the desired relation was an attribute. The expert viewed this inaccuracy as minor. In contrast, in the example of row 5, the expert found the extracted aggregation conceptually wrong, since the desired relation was a generalization.

In response to Q3 (from the questionnaire of Figure 5.9), the expert identified 13 missing relations. In six out of these 13 cases, we could automatically extract the missing relation from other requirements statements in our random sample. From the column chart provided for relevance in Figure 5.10(a), we see that we have a total of 76 (71+5) relevant and maybe relevant relations that are automatically extracted. This means that our approach automatically retrieved  $76/(76+7) \approx 92\%$  of the relevant relations.

Using bootstrapping, similar to that done for Q1 and Q2 in Figure 5.10(b), we obtain the percentage distribution of Figure 5.11 for the retrieved relevant relations. *From this distribution, we obtain a 95% confidence interval of 82%–100% for the percentage of relevant relations that are automatically extracted by our approach.*

**RQ3.** The execution times for our model extraction approach are in the order of minutes over our case study documents (maximum of  $\approx 4$  min for Case B). Given the small execution times observed, we expect our approach to scale to larger requirements documents. Execution times were measured on a laptop with a 2.3 GHz CPU and 8GB of memory.

### 5.5.3 Limitations and Validity Considerations

Internal, construct, and external validity are the validity factors most pertinent to our empirical evaluation. With regard to internal validity, we note that our interview considered the correctness and relevance of extracted relations only in the context of individual requirements statements. We did not present to the expert the entire extracted model during the interview. This raises the possibility that the expert might have made different decisions, e.g., regarding the level of abstraction of the domain model, had he been presented with the entire extracted model. We chose to base our evaluation on individual requirements statements, because using the entire extracted model would have introduced confounding factors, primarily due to layout and information overload issues. Addressing these issues, while important, is outside the scope of our current evaluation, whose primary goal was to develop insights about the effectiveness of NLP for domain model extraction. To ascertain the quality of the feedback obtained from the expert, we covered a reasonably large number of requirements (50 requirements, representing nearly a third of Case A) in our interview, and cross-checked the expert's responses for consistency based on the similarities and analogies that existed between the different relations examined.

With regard to construct validity, we note that our evaluation did not include metrics for measuring

the amount of tacit expert knowledge which is necessary for building a domain model, but which is absent from the textual content of the requirements. This limitation does not pose a threat to construct validity, but is important to point out in order to clarify the scope of our current evaluation. Building insights about the amount of tacit information that needs to be manually added to the domain model and is inherently impossible to obtain automatically requires further studies.

Finally, with regard to external validity, while our evaluation was performed in a representative industrial setting, additional case studies will be essential in the future.

## 5.6 Conclusion

We presented an automated approach based on Natural Language Processing for extracting domain models from unrestricted requirements. As a part of our contributions, we extend the existing set of model extraction heuristics and employ techniques from Information Retrieval domain for domain model extraction. We provided an evaluation of our approach, contributing insights to the as yet limited knowledge about the effectiveness of model extraction in industrial settings.

A key finding from our evaluation is that a substantial proportion of the extracted relations are not relevant to the domain model, although most of these relations are meaningful. Improving relevance is a challenge that needs to be tackled in future work. In particular, additional studies are necessary to examine whether our observations about relevance are replicable. If so, technical improvements need to be made for increasing relevance. To this end, a key factor to consider is that what is relevant and what is not ultimately depends on the context, e.g., what is the intended level of abstraction, and on the working assumptions, e.g., what is considered to be in the scope of a system and what is not. This information is often tacit and not automatically inferable. Increasing relevance therefore requires a human-in-the-loop strategy, enabling experts to explicate their tacit knowledge. We believe that such a strategy would work best if it is incremental, meaning that the experts can provide their input in a series of steps and in tandem with reviewing the automatically-extracted results. In this way, once a piece of tacit knowledge has been made explicit, it can be used not only for resolving incompleteness in the domain model but also for guiding the future actions of the model extractor.

# Chapter 6

## Inter-Requirements Change Impact Analysis

Handling change is an essential part of Requirements Engineering (RE). In early requirements stages, the functions and characteristics of a proposed system may not be adequately known. Early requirements may thus change rapidly as knowledge about the system grows. Once the requirements mature, various other triggers for requirements change may take hold, such as new and evolving user needs.

When a requirement undergoes some change, it is important to be able to analyze how this change impacts other requirements. We refer to this activity as *inter-requirement change impact analysis*. This type of analysis is necessary for maintaining the correctness and consistency of requirements, and is further a prerequisite for analyzing the impact of requirements changes on lower-stream artifacts that are traceable to the requirements, e.g., system design and source code [Jönsson and Lindvall, 2005]. A manual analysis of how a change to one requirement impacts other requirements is time-consuming and presents a challenge for large requirements specifications.

In this chapter, we propose an automated approach based on Natural Language Processing (NLP) for analyzing the impact of change in Natural Language (NL) requirements. Our focus on NL requirements is motivated by the prevalent use of these requirements, particularly in industry. Our approach automatically detects and takes into account the phrasal structure of requirements statements. We argue about the importance of capturing the conditions under which change should propagate to enable more accurate change impact analysis. We propose a quantitative measure for calculating how likely a requirements statement is to be impacted by a change under given conditions. We conduct an evaluation of our approach by applying it to 14 change scenarios from two industrial case studies.

**Structure.** Section 6.1 motivates the study of inter-requirements change impact analysis problem and outlines our contributions in this respect. Section 6.2 presents an approach overview. Section 6.3 provides background information. Sections 6.4 through 6.7 describe the technical components of our approach. Section 6.8 outlines tool support. Sections 6.9 discusses the evaluation of our approach. Section 6.10 compares the approach with related work. Section 6.11 concludes the chapter with a summary and directions for future work.

**R1:** *The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.*

**R2:** *The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk via FTP.*

**R3:** *The mission operation controller shall transmit any detected anomalies to the user help desk.*

**R4:** ~~*The mission operation controller shall implement a configuration management database.*~~

**R5:** *The satellite management system shall provide a mechanism for updating user-defined parameters in the configuration database.*

**R6:** *The satellite management system shall authorise all updates to the telemetry configuration of a satellite before applying the changes to the satellite telemetry database.*

**Figure 6.1.** Example requirements from a satellite control system (with changes). Added text is green and underlined. Removed text is red and struck through.

## 6.1 Motivation and Contributions

We use the example of Figure 6.1 to illustrate how change propagates in NL requirements. The requirements in this example have been drawn from a larger requirements specification for a satellite system, and altered to preserve confidentiality and facilitate illustration.

Suppose R1 is modified as shown, i.e., by replacing “help desk” with “document repository”. The modification is merely a syntactic manifestation of the change. To properly analyze this change, one needs to consider the semantic unit(s) – primarily the phrase(s) in the statement – affected by the modification. Specifically, the change in R1 may not be meant at replacing “help desk” with “document repository”, but rather to replace the noun phrase “user help desk” with “user document repository”.

Another important factor to consider is that a change per se may be inadequate for determining the impact of that change. For example, the change in R1 may be explained in various ways, with each explanation leading to a different impact result. Some possible explanations are: (1) We want to globally rename “user help desk”; in this case the change in R1 affects R2 and R3. (2) We want to avoid communication between “mission operation controller” and “user help desk”; in this case, R3 is affected (the system agent being “mission operation controller”), but not R2. (3) We no longer want to “transmit satellite status reports” to “user help desk” but instead to “user document repository”; in this case, the change in R1 does not affect other requirements. To meaningfully analyze the impact of a change, we need to be able to describe in a precise and yet practical way the conditions under which the change should propagate.

For the above change to R1 and the explanations considered, one can determine the change impact by finding, in other requirements, exact matches for the phrases involved in the change and its possible explanations. Change may further propagate through semantically-related phrases that are not exact matches or close syntactic variations. To illustrate, consider the change in R2, i.e., the addition of

“via FTP”. If this change is meant to indicate that all transfers to the user help desk shall be done via FTP, then the change is very likely to propagate to R1 (in its original unchanged form). Although the process verb used in R1 is “transmit” and not “transfer”, the semantic relatedness between the two verbs needs to be taken into consideration for identifying the impact of change.

To increase the precision of change impact analysis, one further needs to account for the phrasal structure of requirements statements when defining relatedness. To illustrate, suppose that R4 is being removed, the explanation being that a configuration management database is unnecessary. The phrase “configuration management database” only appears in R4. However, the individual words that make up this phrase have matches in R5 and R6. In R5, “configuration” and “database” both appear in the phrase “configuration database”; and “management” appears in “satellite management system”. In R6, “configuration” appears in “telemetry configuration”; “management” appears in “satellite management system”; and “database” appears in “satellite telemetry database”. If one applies a mechanism based on individual words to determine how the removal of R4 impacts other requirements statements, R5 and R6 would be equally likely to be impacted. However, at the phrase level, R5 has a closer match, namely “configuration database”, for the phrase of interest (“configuration management database”). To differentiate R5 and R6 in terms of the likelihood of impact, we need a relatedness measure that takes phrases into consideration.

**Contributions.** We propose an approach for inter-requirement change impact analysis over NL requirements. In so doing, we address several questions highlighted through the example of Figure 6.1: How can we (automatically) identify the phrases in a set of requirements statements? How can we capture change at the level of phrases (and not words)? How can we express the conditions for change propagation? How can we utilize the phrasal structure of requirements to predict the impact of change? And, how can we quantify the likelihood of a requirements statement being impacted by a change?

The core enabling technology for our approach is *Natural Language Processing (NLP)*. We apply a scalable NLP technique, called text chunking [Jurafsky and Martin, 2009], for extracting phrases from requirements. We use the resulting phrases as a basis for detecting change, specifying how change should propagate, and calculating likelihoods for change impact. We implement our approach in a prototype tool. We evaluate our approach using 14 change scenarios from two industrial case studies. The results suggest that our approach is accurate and practical.

While our approach can be applied wherever NL is used for expressing the requirements, the approach is strongly motivated by the situation where one has access to no reliable information other than the requirements’ textual content. This situation occurs, for example, when time and cost pressures prevent the development team from building requirements models, specifying the glossary terms, or capturing the requirements dependencies in a precise manner. Our approach works directly on the text of the requirements and can thus be applied in the situation described above. The main novelty of our work is in using the phrasal structure of requirements to compensate as much as possible for the absence of models, glossaries, and dependency links.

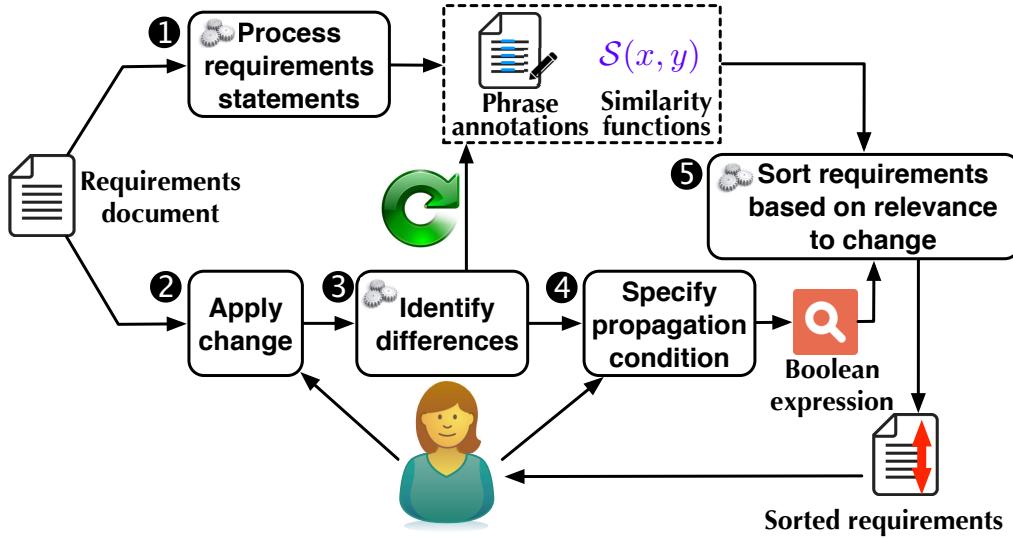


Figure 6.2. Approach overview.

## 6.2 Overview of the Approach

The process in Figure 6.2 presents an overview of our approach. The process takes as input a requirements document comprised of NL requirements statements. In Step 1, *Process requirements statements*, we apply NLP to automatically (1) identify the constituent phrases of the requirements statements and (2) compute pairwise similarity scores for all tokens (words) that appear in the identified phrases. The outputs from Step 1 are annotations delineating the phrases in the statements, and similarity functions capturing the syntactic and semantic similarities between the tokens. In Section 6.4, we elaborate how phrase detection and similarity calculation is performed.

In Step 2, *Apply change*, the user makes a change to the requirements document. We discuss this step in Section 6.5. Our approach lifts proposed changes from the level of tokens to the level of phrases. This is performed in Step 3, *Identify differences*, discussed alongside Step 2 in Section 6.5. If the change introduces phrases with words that did not exist in the requirements document before, the similarity functions produced in Step 1 will be accordingly updated.

As we argued earlier, a change by itself may not provide enough information to accurately analyze the impact of the change. In Step 4, *Specify propagation condition*, the user captures the desired condition under which the change should propagate. The propagation condition is specified in terms of phrases, using a boolean expression. When specifying the propagation condition, the user has access to all the (automatically-detected) phrases in the original requirements document as well as any added or deleted phrases detected by Step 3. We discuss propagation conditions in Section 6.6. Further and as part of our tool support (Section 6.8), we provide a user interface to facilitate writing these conditions.

In Step 5, *Sort requirements based on relevance to change*, the requirements statements are ordered based on the likelihood of being impacted by the change. The ordering is derived from a quantitative matching of the change propagation condition (from Step 4) against the statements. The matching is computed using the outputs of Step 1. As we elaborate in Section 6.7, we consider the phrasal structure of requirements statements when computing a matching.

The output from Step 5 is displayed to the user. Ideally, we would like the ordering to partition the requirements statements, with all impacted statements appearing at the beginning and all not-impacted ones at the end of the ordered set. This would help users focus on top-ranked statements with a higher likelihood of undergoing change. In Section 6.9, we evaluate how close our approach is to the ideal case.

We note three considerations about the process of Figure 6.2: First, the process is iterative; the user can apply and analyze changes in a consecutive manner by returning to Step 2 (*Apply change*). Second, the process is interactive. This in particular requires the flexibility to make changes in real-time and obtain impact results with no, or only short, delays. Subsequently, computationally-intensive tasks, such as the calculation of similarity scores, need to be minimized over the interactive path in the process, i.e., Steps 2 through 5. The aim of Step 1 in the process is to factor out as much of these computations as possible from the interaction path. Specifically, one can run Step 1 offline and prior to change impact analysis. If a change warrants updating the outputs of Step 1, the updates are made incrementally in Step 3, as already discussed. Third, the approach allows the user to skip Steps 2–3, and start directly with articulating the change propagation condition in Step 4. This is useful when one wants to hypothesize and analyze the impact of a proposed change, say, deleting all requirements statements concerned with processing a specific object, before deciding whether or not to make the change.

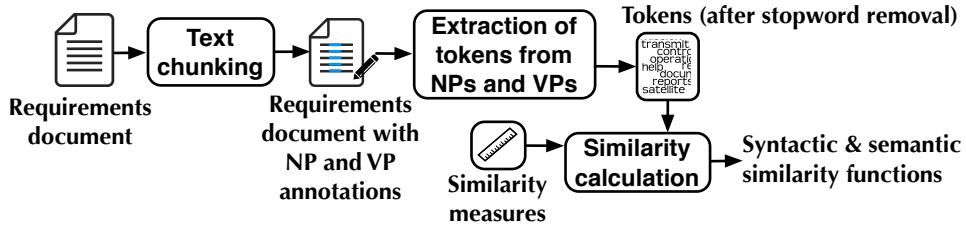
## 6.3 Background

In this section, we introduce the NLP techniques that are used in this chapter: (1) detection of phrases in sentences and (2) calculation of similarity measures.

**Phrase detection.** We use *text chunking* for automatic detection of phrases. We provided the necessary background on text chunking in Chapter 3 (see Section 3.2.2). As we discussed in this previous chapter, several alternative chunkers are available within existing NLP toolkits. We evaluated the accuracy of some of these alternatives over requirements documents in Chapter 3. In this current chapter, we use the OpenNLP chunker [OpenNLP, 2016], which, based on the evaluation of Chapter 3, is one of the most robust alternatives.

For change impact analysis, we are interested in the Noun Phrases (NPs) and Verb Phrases (VPs) as the main meaning-bearing elements of sentences. We enhance the results of text chunking with heuristics that merge adjacent NPs under certain conditions. These heuristics, as discussed previously in Chapter 4 (see Table 4.4), are aimed at maintaining the semantic link between closely related NPs. For example, chunking would decompose “configuration of a satellite” into two NPs: “configuration” and “a satellite”. If we treat these NPs separately, the context for “configuration” will be lost. To address this issue, we merge into a single NP any pair of adjacent NPs that match one of the following patterns: *NP of NP*, *NP’s NP*, and *NP in NP*.

**Similarity measures.** Similarity measures for NL can be *syntactic* or *semantic*. Syntactic measures compute similarity scores based on the string content of text segments, sometimes combined with frequencies. An example syntactic similarity measure is *Levenshtein similarity* [Manning et al., 2008],



**Figure 6.3.** Details of the requirements processing step.

which computes a similarity between two strings based on the minimum number of character edits required to transform one string into the other.

Semantic measures are calculated based on correlations captured in dictionaries. An example semantic similarity measure is the *Path measure* [Rus et al., 2013], which computes a similarity score between two words based on the shortest path between them in an *is-a* hierarchy (e.g., a “car” *is-a* “vehicle” and so is a “scooter”).

Most similarity measures, both syntactic and semantic, are normalized to produce a value between 0 and 1, with 0 signifying no similarity and 1 signifying a perfect match. Since there are numerous similarity measures to choose from, it is important to investigate through empirical means which measures yield the best results for a specific task.

Syntactic measures are generally best-suited for matching variations of the same word or phrase, e.g., “components of the system” and “system components”, and for dealing with words that are misspelled or not in dictionaries. Semantic measures are most suitable for matching words that are syntactically different but have closely-related meanings, e.g., “message” and “communication”. Semantic measures further provide an accurate basis for dealing with language morphology, for instance, nominalizations, e.g., “handling” versus “handle”. Due to the complementary characteristics of syntactic and semantic measures, these two classes of measures may be combined to produce higher-quality similarity scores [Nejati et al., 2012].

We note that semantic measures are typically more expensive than syntactic measures to compute. Therefore, when using semantic measures, one needs to consider the level of scalability required for the task at hand. Recent advances in NLP address many of the scalability challenges associated with semantic measures. In particular, the newly-developed SEMILAR (SEMantic simILARity) toolkit [Rus et al., 2013] provides efficient implementations for a number of semantic measures.

In this chapter, we experiment with several syntactic and semantic measures, and their combinations for change impact analysis. The best measures for our application context are discussed in our empirical evaluation (Section 6.9).

## 6.4 Processing of Requirements

The requirements processing step, the details of which are depicted in the diagram of Figure 6.4, detects the phrases in the requirements statements, extracts the tokens of these phrases, and computes similarity scores for the extracted tokens.

Requirements phrases are identified using text chunking, as we described in Section 6.3. From

the annotation produced by text chunking, we use only the noun and verb phrases (NPs and VPs). As noted earlier, these phrases are the most important to the meaning of sentences. Following text chunking, the identified NPs and VPs are broken down into their tokens to create a global, non-redundant set of tokens. We discard stopwords from this set. Stopwords are tokens that appear so frequently in the text that they lose their usefulness for text processing [Manning and Schütze, 1999]. Stopwords include but are not limited to determiners, predeterminers, pronouns, conjunctions, and prepositions. We adapt the stopwords from the Brown corpus [Francis and Kucera, 1982]. For example, “the user help desk” in R1 (Figure 6.1) would be reduced to the following tokens: “user”, “help”, “desk”; “maintenance and service plans” in R2 (Figure 6.1) would be reduced to: “maintenance”, “service”, and “plans”.

Let  $T$  denote the union of all tokens extracted from the NPs and VPs of a requirements document, with the stopwords removed. We subject  $T$  to similarity calculation, as shown in Figure 6.4. A (token-level) similarity function is a total function  $T \times T \rightarrow [0..1]$ , assigning a normalized value to every pair  $(t, t') \in T \times T$  of tokens. The closer the similarity score is to 1, the more similar a pair of tokens are with respect to the similarity measure being used. We consider three alternative strategies for building a similarity function:

1. *syntactic only*, where a similarity function,  $\mathcal{S}_{syn}$ , is calculated using a *syntactic* measure, e.g., Levenshtein similarity;
2. *semantic only*, where a similarity function,  $\mathcal{S}_{sem}$ , is calculated using a *semantic* measure, e.g., the path measure;
3. *combined*, where, for every pair  $(t, t')$  of tokens, we take  $\max(\mathcal{S}_{syn}(t, t'), \mathcal{S}_{sem}(t, t'))$ . Using  $\max$ . is motivated by the complementarity of syntactic and similarity measures [Nejati et al., 2012].

We zero out token similarity scores smaller than 0.3 to exclude poor token matches from further analysis. Applying this threshold is common practice when two bags of tokens are being compared based on their pairwise token similarities, e.g., see [Rus et al., 2013]. In Section 6.7, we use token similarities alongside phrasal information for predicting change impact. We discuss in Section 6.9 which strategy from the three above and which similarity measures yield the most accurate results.

## 6.5 Change Application and Differencing

Applying a change (Step 2 of the process of Figure 6.2) is an interactive step where the user *adds*, *deletes* or *updates* a requirements statement. To enable phrase-level analysis of changes, we cast these change operations as additions and deletions of phrases (NPs and VPs). Specifically:

- Adding (resp., deleting) a requirement amounts to adding (resp., deleting) a collection of phrases. For example, deleting R4 shown in Figure 6.1 is treated as deleting the VP “shall implement” and deleting the NPs “the mission operation controller” and “a configuration management database”.
- Updating a requirement amounts to a combination of phrase additions and deletions. For example, consider the requirement in Figure 6.4, which is a slight extension of R1 in Figure 6.1. The textual change here is deleting “help desk” and adding “document repository at a local server”. This update is treated as deleting “the user help desk”, and adding “the user document repository” and “a local server”.

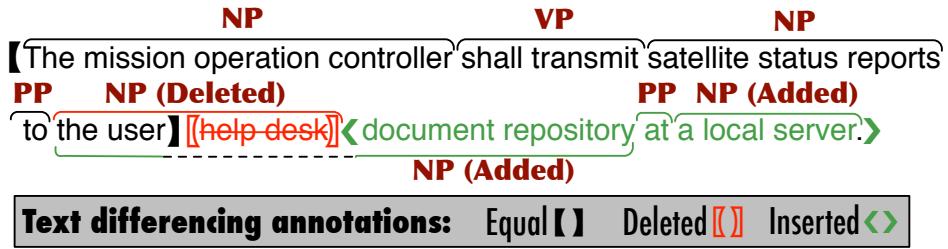


Figure 6.4. Differencing example.

**Input:** The original requirement,  $R_{old}$ , and the changed one,  $R_{new}$ ;

**Output:** Set of phrase additions and deletions;

```

1: Let  $P_{old}$  be the set of phrases in  $R_{old}$ , and  $P_{new}$  that in  $R_{new}$ ;
2: if  $P_{old} = \emptyset$  then return  $\{(p, Added) \mid p \in P_{new}\}$ ;
3: else if  $P_{new} = \emptyset$  then return  $\{(p, Deleted) \mid p \in P_{old}\}$ ;
4: end if
5: Let  $\ell = \emptyset$ ; /* Initialize the set of phrases to be returned. */
6: for all  $region \in \text{diff-match-patch}(R_{old}, R_{new})$  do
7:   if  $region.type = Equal$  then do nothing;
8:   else if  $region.type = Deleted$  then
9:      $\ell := \ell \cup \{(p, Deleted) \mid (p \in P_{old}) \wedge (p \text{ overlaps with } region)\}$ ;
10:  else if  $region.type = Inserted$  then
11:     $\ell := \ell \cup \{(p, Added) \mid (p \in P_{new}) \wedge (p \text{ overlaps with } region)\}$ ;
12:  end if
13: end for
14: return  $\ell$ 
```

Figure 6.5. Algorithm for detecting added & deleted phrases (phrasal differencing).

The algorithm of Figure 6.5 outlines the procedure we apply for automatically detecting added and deleted phrases. The algorithm superimposes the NP and VP annotations obtained from text chunking over differencing annotations obtained from a standard text differencing tool, e.g., *diff-match-patch* [Fraser, 2012]. Text differencing partitions a given text (in our case, a requirements statement) into regions annotated with *Equal*, *Deleted*, and *Inserted*. These annotations, illustrated in Figure 6.4, denote unchanged, deleted, and inserted text segments, respectively. Given the text chunking and differencing annotations, the algorithm analyzes the overlaps between the phrases and the changed text regions, returning a set of tuples of the form  $(p, Op)$  where  $p$  is a phrase and  $Op$  is either *Added* or *Deleted*.

For example, the output from the algorithm of Figure 6.5 over the requirements change shown in Figure 6.4 is the following set:  $\{("the user help desk", Deleted), ("the user document repository", Added), ("a local server", Added)\}$ . These added and deleted phrases can be used by requirements analysts for specifying the change propagation condition, discussed next.

## 6.6 Specification of Propagation Condition

The algorithm of Figure 6.5 identifies what has changed but does not explain the context and circumstances around the change. As we illustrated in Section 6.1, an accurate analysis of the impact of a change may not be possible based solely on the change itself. To obtain meaningful results through automation, the analyst needs to make explicit any known criteria that the impacted requirements are

1	$\langle \text{expression} \rangle$	$::= \langle \text{composite-expr} \rangle \mid \langle \text{atomic-expr} \rangle$
2	$\langle \text{composite-expr} \rangle$	$::= (" \langle \text{expression} \rangle \text{ AND } \langle \text{expression} \rangle ") \mid (" \langle \text{expression} \rangle \text{ OR } \langle \text{expression} \rangle ")$
3	$\langle \text{atomic-expr} \rangle$	$::= \langle \text{phrase} \rangle \mid \langle \text{verbatim-text} \rangle$
4	$\langle \text{phrase} \rangle$	$::= \langle \text{pos-phrase} \rangle \mid \langle \text{neg-phrase} \rangle$
5	$\langle \text{verbatim-text} \rangle$	$::= \langle \text{pos-verbatim-text} \rangle \mid \langle \text{neg-verbatim-text} \rangle$
6	$\langle \text{pos-phrase} \rangle$	$::= \langle \text{PHRASE} \rangle$
7	$\langle \text{neg-phrase} \rangle$	$::= \text{NOT } \langle \text{pos-phrase} \rangle$
8	$\langle \text{pos-verbatim-text} \rangle$	$::= [" \langle \text{TEXT} \rangle "]$
9	$\langle \text{neg-verbatim-text} \rangle$	$::= \text{NOT } \langle \text{pos-verbatim-text} \rangle$

**Figure 6.6.** Grammar for propagation conditions.

**Table 6.1.** Example propagation conditions and their explanation.

Req. (from Figure 6.1)	Explanation	Propagation Condition
1	Replace user help desk in subsystems X and Y only.	user help desk AND ([X] OR [Y])
2	Replace user help desk unless it is for subsystem X.	user help desk AND (NOT [X])
3	R1	Avoid communication between mission operation controller and user help desk.
4		mission operation controller AND user help desk AND transmit
5	R2	Do not transmit satellite status reports to user help desk.
		satellite status report AND user help desk AND transmit
6	R4	We want to do all transfers to the user help desk via FTP.
		transfer AND user help desk
		configuration database management
		management

expected to meet. Stated otherwise, for a given change, the analyst must provide a condition characterizing the requirements that the change is likely to propagate to. We refer to this condition as the *propagation condition*. Essentially, the propagation condition is the analyst's answer to the following: *What phrases do you expect to see or not to see in the requirements impacted by the change?*

We use a restricted form of boolean expressions, shown in the grammar of Figure 6.6, for capturing propagation conditions. An expression is either composite or atomic (L. 1). Composite expressions are built recursively using `AND` and `OR` (L. 2). Atomic expressions can be phrases or verbatim text (L. 3). The verbatim text option is provided to support exact string search, as implemented in text editors. When phrases are indicated, a quantitative measure is applied for search to account for the syntactic and semantic variations of phrases and how the constituent words of the phrases appear in the requirements (Section 6.7). Phrases and verbatim text can both be negated to state they must be absent (L. 4-5). The symbols  $\langle \text{PHRASE} \rangle$  and  $\langle \text{TEXT} \rangle$  (L. 6, 8) are terminals. Verbatim text is enclosed in brackets (L. 8) to distinguish it from phrases (L. 6).

Table 6.1 provides examples of propagation conditions, based on the changes made to R1, R2, and R4 in the requirements of Figure 6.1. In each case, we show the requirements statement number, a possible explanation for the change, and the propagation condition. In rows 1-4 and 6 of Table 6.1, the change, as detected by the algorithm of Figure 6.5, plays a role in defining the propagation condition; whereas in row 5, the newly-added phrase ("FTP") does not appear in the propagation condition. This is because the propagation condition is meant to characterize what is likely to be seen in the impacted requirements, hence "FTP" not being part of the condition in row 5.

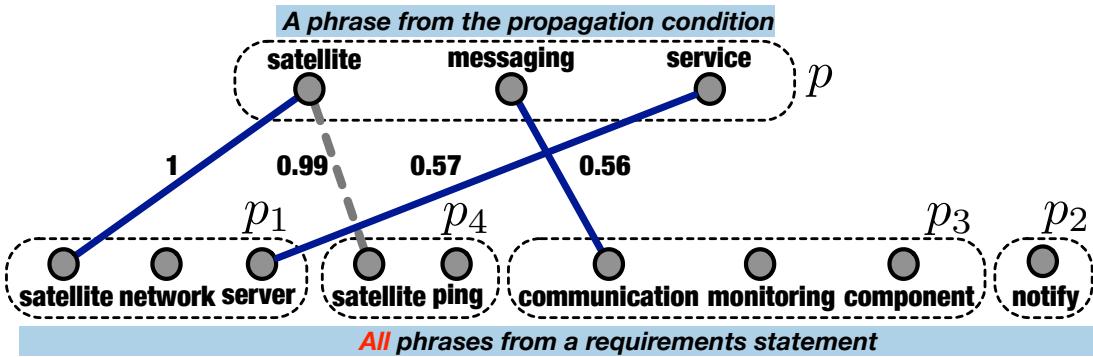


Figure 6.7. Matching a propagation condition phrase against a requirement.

Since one cannot in general assume that analysts are comfortable with writing logical expressions, it is important to provide a more intuitive layer over logical expressions for specifying propagation conditions. To this end, we draw on findings in the Information Retrieval community, where it has been observed that people often conceptualize search queries in terms of *Conjunctive Normal Form (CNF)* expressions [Pirkola et al., 1999]. While our approach supports the grammar of Figure 6.6 without restrictions, our tool support (Section 6.8) limits propagation conditions to CNF expressions. This enables us to shield the user from logical expressions through a user interface. All the propagation conditions in Table 6.1 are in CNF.

As the examples in Table 6.1 show, propagation conditions are associated closely with *why* a certain change is being made. In our approach, we elicit propagation conditions directly from the analyst, rather than having the analyst first fully conceptualize why they made a change and then attempting to derive the propagation condition from their answer. Our decision is motivated by our experience, indicating that practitioners find it more difficult to answer why they made a change than to provide clues about how they would propagate it. This observation relates to known cognitive limitations of answering “why” questions about requirements and design rationale [Dutoit et al., 2006].

## 6.7 Calculation of Impact Likelihoods

Given a propagation condition  $\varphi$ , we calculate for every requirements statement  $R$  a normalized matching score  $\mathcal{M}(\varphi, R)$ . The score is a measure of how likely  $R$  is to be impacted by the change associated with  $\varphi$ . The higher the score, the more likely  $R$  is to be impacted. The matching score is computed for all requirements statements in a specification. The statements are then sorted in descending order of the score and presented to the analyst. Below, we explain how the matching scores are computed. We discuss how to use the resulting sorted list in Section 6.9 (RQ2 and RQ3).

The matching score is computed bottom-up, as per the grammar of Figure 6.6, from atomic to composite expressions. For positive verbatim text  $v$ , we evaluate  $\mathcal{M}(v, R)$  to 1 if  $R$  contains  $v$ ; and 0, otherwise. The core part of the matching is calculating scores for positive phrases. For positive phrase  $p$ , we calculate  $\mathcal{M}(p, R)$  according to the following process:

(1) Let  $\llbracket t_1, \dots, t_n \rrbracket$  be the bag of  $p$ ’s tokens. Let  $p_1, \dots, p_k$  be the phrases in  $R$ ; and let  $\llbracket t'_1, \dots, t'_m \rrbracket$  be the bag of the tokens of these phrases. Note that stopwords are removed from both bags. Construct a bipartite graph  $G$ , where the nodes of the first and the second parts are  $\llbracket t_1, \dots, t_n \rrbracket$  and

$\llbracket t'_1, \dots, t'_m \rrbracket$ , respectively. To illustrate, suppose  $p$  is “the satellite matching service” and  $R$  is “The satellite network server shall notify the communication monitoring component after each satellite ping.” The phrases in  $R$  are “The satellite network server”, “shall notify”, “the communication monitoring component”, and “each satellite ping”. In Figure 6.7, we show the nodes of  $G$ . The phrases of  $R$  (and thus their constituent tokens) are ordered in a certain manner. We discuss this ordering and its use in step (2) below.

(2) We connect every pair  $(t, t') \in \llbracket t_1, \dots, t_n \rrbracket \times \llbracket t'_1, \dots, t'_m \rrbracket$  with a weighted edge. The weight is assigned by one of the token similarity calculation strategies discussed in Section 6.4. Which strategy is the most accurate is discussed in Section 6.9 (RQ1). With the edges added to  $G$ , we obtain a full weighted bipartite graph [Cormen et al., 2009]. We use this graph for finding an optimal matching between the tokens of  $p$  and those of  $R$ . The optimization is cast into the assignment problem [Cormen et al., 2009]. Solving the assignment problem yields a set of edges such that no two edges share the same token as an endpoint, while maximizing the sum of the weights of the selected edges. In Figure 6.7, the edges of the optimal matching are shown using solid lines. The number next to each edge denotes the edge’s weight. For readability reasons, we do not show  $G$ ’s entire edge set.

Before attempting to find an optimal match, we adjust the weights of the edges in  $G$  to nudge matching towards using the smallest possible number of phrases from  $R$ . To illustrate, consider the term “satellite” in  $p$  (Figure 6.7), for which there are two exact matches in  $R$ , i.e., in phrases  $p_1$  and  $p_4$ . Despite both  $p_1$  and  $p_4$  containing matches, picking “satellite” from  $p_1$  is more sensible because at the phrase level,  $p_1$  is a closer match to  $p$  than  $p_4$  is. To guide matching to pick “satellite” from  $p_1$ , we rank the phrases in  $R$  according to their phrase-level similarity with  $p$ . We then levy a small penalty for picking tokens from phrases with a higher rank. More precisely, the weight of an edge between tokens  $t$  and  $t'$  is adjusted as follows:  $\text{adjusted weight} = (1 - (r-1)/100) \times \text{original weight}$ , where  $r$  is the rank of the phrase in which  $t'$  is located. We use a syntactic similarity measure applied to phrases to compute the ranking. Finding the best such measure for ranking is addressed in Section 6.9 (RQ1). For instance, with Levenshtein similarity applied for ranking, the phrases of  $R$  (Figure 6.7) in descending order of similarity with  $p$  are  $p_1, p_4, p_3$ , and  $p_2$ . The dashed edge in Figure 6.7, which is not part of the optimal matching, has an adjusted weight of  $0.99 = (1 - (2-1)/100)$ , noting that the rank of  $p_4$  is 2.

(3) We calculate the matching score for  $p$  as follows:

$$\mathcal{M}(p, R) = 2 \times \frac{\text{sum of the weights of edges in optimal matching}}{N_1 + N_2 + 0.5 \times (N_3 - 1)}$$

where  $N_1$  is the number of tokens in  $p$ ,  $N_2$  is the number of matched tokens from  $R$ , and  $N_3$  is the number of phrases from  $R$  involved in the optimal matching. For the example of Figure 6.7, the calculations are as follows:  $\text{sum of weights} = 1.0 + 0.56 + 0.57 = 2.13$ ,  $N_1 = 3$ ,  $N_2 = 3$ , and  $N_3 = 2$  because two phrases ( $p_1$  and  $p_3$ ) contribute tokens to the optimal matching. Hence,  $\mathcal{M}(p, R) = (2 \times 2.13) / (3 + 3 + 0.5(2-1)) = 0.66$ .

The above formula is a modification of a commonly-used formula for comparing name labels [Nejati et al., 2012]. In our modified formula,  $N_2$  accounts for only the matched tokens of  $R$  rather than

Original Req: A WASP application shall be able to establish a phone call connection using the 3G Platform via the WASP platform .

Changed Req: A WASP application shall be able to establish a phone call connection and a video chat connection using the 3G Platform via the WASP platform .

Changes: ADDED NP @ R26.video chat connection

**Drag and Drop**

ANALYSIS: The search results show that the requirement has been modified to include a 'video chat connection'. The '3G Platform' term is highlighted with a red dashed border, indicating it is the new component being added. The 'WASP application' and 'WASP platform' terms are also present, maintaining the original context.

((( WASP application OR WASP platform ) AND ( 3G Platform ) AND ( connection )))

ID	Requirements Statement	Score
R26	A WASP application shall be able to establish a phone call connection using the 3G Platform via the WASP platform .	1
R22	The WASP platform should be able to use several communication services offered by the 3G platform .	0.77

**Figure 6.8.** NARCIA's user interface for (a) specifying propagation conditions, and (b) reviewing change impact analysis results (tool's output has been truncated).

all its tokens. This modification prevents the matching scores from being affected by the length of requirements statements. Our modified formula further introduces an additional factor of  $0.5 \times (N_3 - 1)$  in the denominator. This additional factor is motivated by the intuition that a smaller number of participating phrases from  $R$  in the optimal match makes  $R$  likely to have a stronger relationship to  $p$ . For instance, had  $p_1$  in Figure 6.7 been “satellite communication server”, the optimal match would have used only  $p_1$  from  $R$ . This would have been rewarded by increasing the matching score from 0.66 to  $(2 \times 2.13) / (3 + 3 + 0.5(1 - 1)) = 0.71$ .

For negated atomic and composite expressions in the grammar of Figure 6.6, we calculate the matching scores as follows:

- $\mathcal{M}(\text{NOT } z, R) = 1 - \mathcal{M}(z, R)$ ;  $z$  is a phrase or verbatim text
- $\mathcal{M}(\varphi_1 \text{ AND } \varphi_2, R) = \mathcal{M}(\varphi_1, R) \times \mathcal{M}(\varphi_2, R)$
- $\mathcal{M}(\varphi_1 \text{ OR } \varphi_2, R) = \max(\mathcal{M}(\varphi_1, R), \mathcal{M}(\varphi_2, R))$

## 6.8 Tool Support

We have implemented our approach in a prototype tool, NARCIA (NAtural language Requirements Change Impact Analyzer). In Figure 6.8, we present two screenshots of the tool. Figure 6.8(a) shows the interface for specifying propagation conditions. The tool restricts these conditions to CNFs, as discussed in Section 6.6. Each box marked with a  $\star$  on the screenshot of Figure 6.8(a) is a CNF clause, i.e., a disjunction of phrases and possibly verbatim text items. The user can define as many clauses as necessary using the “Add” button.

To assist users in writing propagation conditions, the tool presents the original and changed requirements statements alongside the added and deleted phrases detected via phrasal differencing. The user can drag and drop any phrase from this information into the clause boxes, as illustrated in Figure 6.8(a). For convenience, the tool automatically removes determiners and predeterminers from phrases. The drop-down lists in the clause boxes are populated with the phrases of the underlying requirements document to allow easier access to these phrases. In addition, the user has the option of directly typing in phrases or verbatim text into the clause boxes. Once the propagation condition has been provided and the “Analyze” button pressed, the tool produces a sorted list of requirements statements based on how well the statements match the provided condition. Figure 6.8(b) shows a (truncated) example of the tool’s output. The propagation condition is shown on the top of the result page as a logical expression.

NARCIA is implemented in Java and is approximately 8K lines of code, excluding comments and third-party libraries. For more details, see: <https://sites.google.com/site/svvnarcia/>.

## 6.9 Evaluation

In this section, we investigate, through two industrial case studies, the following Research Questions (RQs):

**RQ1. Which similarity measures are best suited to our approach?** The choice of similarity measures used for calculating impact likelihoods has a direct bearing on the quality of the results. RQ1 aims to identify the syntactic and semantic similarity measures that lead to the most accurate results.

**RQ2. How should analysts use the sorted requirements list produced by our approach?** For our approach to be useful, analysts need to determine how much of the sorted list is worthwhile inspecting. In other words, they need to determine a point in the list beyond which the remainder of the list is unlikely to contain impacted requirements. The aim of RQ2 is to develop systematic guidelines on how to choose this point.

**RQ3. How effective is our approach?** Assuming that the guidelines resulting from RQ2 are followed, RQ3 aims to determine if our approach can reliably identify the impact set and at the same time save substantial inspection effort.

**RQ4. How scalable is our approach?** RQ4 aims to establish if our approach has a reasonable execution time.

Table 6.2 outlines key information about our case studies. The first case, hereafter called Case-A, concerns a proprietary requirements document for a satellite software component that is under development by our industry partner, SES TechCom. The second case, hereafter called Case-B, is based on a public requirements document for a context-aware mobile service platform [Ebben, 2002]. Both documents represent real systems and were written by practitioners. Data collection for Case-A was performed as part of our current work; whereas, Case-B is drawn from the evaluation material of a previous, and different, impact analysis technique [Goknil et al., 2014a, Goknil et al., 2014b]. For each case, we provide in Table 6.2 a brief description, the number of requirements statements, the number of phrases and distinct tokens, and the number of change scenarios considered. The source material for Case-B is available on our tool’s website at <http://sites.google.com/site/svvnarcia/>.

**Table 6.2.** Case studies used in the evaluation.

Case	Description	# of requirements	# of phrases	# of distinct tokens	# of change scenarios
<b>Case-A</b>	Simulator module for a satellite ground station	160	673	648	9
<b>Case-B</b>	3G mobile service platform	72	267	263	5

**Table 6.3.** Shapes of propagation conditions and sizes of impact sets.

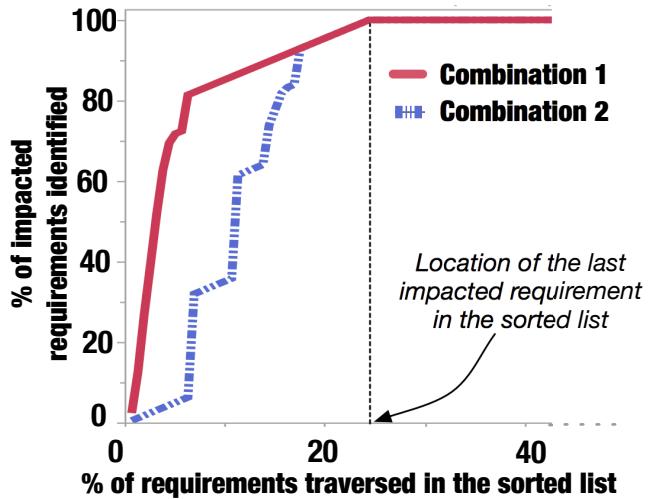
Scenario	Propagation Condition Pattern	Size of Impact Set
Case-A	{NP} AND {NP}	4
	{NP} OR {NP}	8
	{NP}	39
	((NP) OR (NP)) AND {NP}	5
	{NP} OR {NP}	10
	{NP} AND {NP}	3
	{NP} AND {NP}	7
	{NP} OR {NP}	5
	{verbatim-text} AND {NP}	3
Case-B	{NP} AND {NP}	2
	{NP}	9
	{NP} AND {NP} AND {NP}	1
	{NP} AND {NP}	1
	((NP) OR {NP}) AND ((NP) OR {NP})	9

The change scenarios in Case-A are *real* and based on the change history of the underlying document. We identified 14 scenarios, of which we use only 9 in our evaluation due to reasons described in the next paragraph. In Case-B, there are 5 scenarios, which are *hypothetical* but validated with the

experts in Case-B in terms of being meaningful [Goknil et al., 2014b]. In both cases, the impact sets for the changes were provided by the experts involved in writing the requirements. Table 6.3 shows the size of the impact set for each change scenario along with the shape of the propagation condition.

In Case-A, the propagation conditions were specified directly by the lead engineer. Phrasal search was used in only 9 (out of the 14) scenarios. The other 5 scenarios involved only verbatim text search and were consequently excluded due to their limited usefulness in our evaluation. In Case-B, we did not have access to the experts for specifying the propagation conditions. Three researchers independently wrote the conditions and then reached consensus on them. To avoid validity threats, the phrases in the propagation conditions of Case-B were limited to the phrases that appeared in the changed requirements statements (pre- and post-change) as well as the brief description of change rationale available in the existing documentation [Goknil et al., 2014b].

Next, we discuss our RQs based on Case-A and Case-B:



**Figure 6.9.** Accuracy of lists produced by two combinations of similarity measures.

**RQ1.** To answer this RQ, we define a notion of accuracy for the sorted lists produced as described in Section 6.7. We do so using charts that show the percentage of impacted requirements identified (Y-axis) against the percentage of requirements traversed in the list (X-axis). In Figure 6.9, we provide charts for two sorted lists (of the same requirements set), computed by two different combinations of similarity measures.

A simple accuracy metric would be the percentage of requirements traversed in a sorted list to identify *all* impacted requirements. While intuitive, this metric cannot distinguish the combinations in Figure 6.9. In both combinations, all the impacted requirements are identified after traversing 24% of the lists. Nevertheless, Combination 1 is a better alternative as it produces better results *earlier*. To reward earlier detection of impacted requirements, we use the *Area Under the Curve* (AUC) for evaluating accuracy. AUC can tell apart the combinations in Figure 6.9, as the metric is larger for Combination 1.

We instantiated our approach using pairwise combinations of 10 syntactic measures from SimPack [SimPack, 2016] and 9 semantic measures from SEMILAR, including alternatives where only a syntactic or only a semantic measure is applied (i.e., where a constant zero function is used for either semantic or syntactic similarity). This yields  $(10 + 1) \times (9 + 1) - 1 = 109$  combinations. For the phrase ranking stage in the process described in Section 6.7, we used the syntactic similarity measure in a given combination, or Levenshtein similarity when only a semantic measure was being applied for tokens. We ran these combinations on all our change scenarios, calculating the AUCs. Let  $S$  be the sum of the AUCs for the change scenarios in either Case-A or Case-B, resulting from a particular combination. The combination that maximizes  $S$  is deemed the best for the respective case study.

We obtain the best result for Case-A when *Levenshtein similarity* –a syntactic measure– is applied alone, and for Case-B when *Path* –a semantic measure– is applied alone. This discrepancy is explained as follows: In Case-A, the requirements were written by a small group of engineers, and the propagation conditions were specified directly by one of these engineers. In contrast, the requirements in Case-B were written by a consortium of companies and the propagation conditions were written by people other than those involved in writing the requirements. Our analysis indicates that using semantic similarities is important for handling the heterogeneity seen in Case-B. The absolute bests

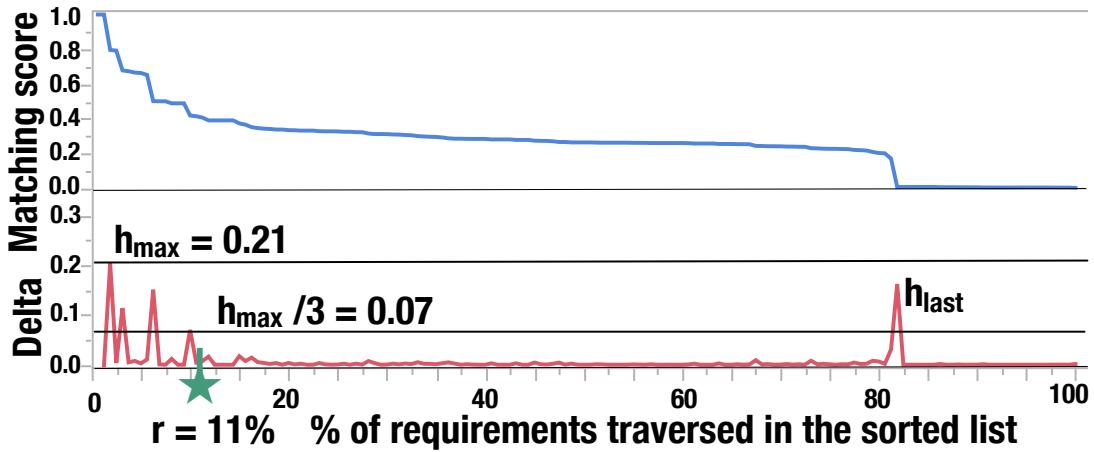


Figure 6.10. Delta chart for identifying the cutoff ( $r$ ).

in Case-A and Case-B are followed closely by the combination of Levenshtein and Path. In Case-A, this combination is distinguished by only 1% from when Levenshtein is applied alone (in terms of the sum of the AUCs); and, in Case-B, the difference between the combination of Levenshtein and Path versus Path alone is only 2%. Given these negligible differences and the complimentary nature of syntactic and semantic measures, *we recommend the combined use of Levenshtein similarity and Path*. These measures were briefly introduced in Section 6.3.

Overall, our recommended combination yields results that, when compared to results from other combinations, are better (in terms of AUC) by an average of 11% in Case-A and 4% in Case-B. The largest accuracy difference between our recommended combination and other combinations was over a change scenario (A.9) in Case-A, where our recommended combination outperformed another by 33%. The remaining RQs are answered based on our recommended combination.

**RQ2.** Outside an evaluation setting, one cannot know *a priori* how much of a sorted list needs to be inspected before all the impacted requirements statements have been seen. To decide how much of a sorted list is worthwhile inspecting, we define a notion of *cutoff*. This notion is defined based on a delta chart, an example of which is shown in Figure 6.10 for one of the change scenarios of Case-A. In the chart, at any position  $i$  on the X-axis, the Y-axis is the difference between the matching scores (impact likelihoods) at positions  $i$  and  $i - 1$ . For easier understanding, we further show, on the top of the figure, the matching scores. We set the cutoff to be the point on the X-axis after which there are no significant peaks in the delta chart. Intuitively, the cutoff is the point beyond which the matching scores no longer adequately distinguish the requirements in terms of being impacted. What constitutes a significant peak is relative. Based on our experimental experience, a peak is significant if it is larger than one-third of the highest peak in the delta chart, denoted  $h_{\max}$  in Figure 6.10. The only exception is the peak caused by zeroing out token similarities smaller than 0.3 (see Section 6.4). This peak, if it exists, is always the last one and hence denoted  $h_{\text{last}}$ . Since  $h_{\text{last}}$  is a mathematical artifact, it is discarded when the cutoff is being determined.

More precisely, we define the cutoff  $r$  to be at the end of the right slope of the last significant peak (excluding  $h_{\text{last}}$ ). In the example of Figure 6.10,  $h_{\max} = 0.21$ . Hence,  $r$  is at the end of the last peak with a height  $> h_{\max}/3 = 0.07$ . *We recommend that analysts should inspect the requirements statements up to the cutoff and no further.* In the example of Figure 6.10, the cutoff is at 11% of the

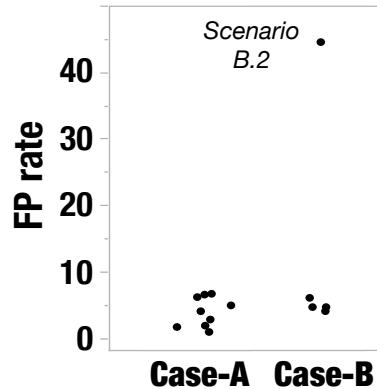


Figure 6.11. FP rates.

sorted list. It is important to note that the cutoff is *automatically computable*. Hence, the delta chart and its interpretation are transparent to the users.

**RQ3.** We answer this RQ based on the cutoff from RQ2. To be effective, our approach must produce a small number of *False Positives (FP)* and *False Negatives (FN)*. An FP is a non-impacted requirements statement that appears before the cutoff and is thus subject to manual inspection. Formally, the set of all FPs is given by  $\mathcal{R}_{\text{inspected}} \setminus \mathcal{R}_{\text{impacted}}$ , where  $\mathcal{R}_{\text{inspected}}$  is the set of inspected requirements (for a specific change) and  $\mathcal{R}_{\text{impacted}}$  is the set of impacted requirements (by that change). An FN is an impacted requirements statement that appears after the cutoff and is thus missed. The set of all FNs for a given change is  $\mathcal{R}_{\text{impacted}} \setminus \mathcal{R}_{\text{inspected}}$ .

Figure 6.11 shows, for each change scenario in Case-A and Case-B, the percentage of FPs in terms of the total number of requirements in the respective case. In Case-A, the FP rate is between 1% and 7%, and in Case-B – between 6% and 8%, except for an outlier (scenario B.2) with an FP rate of ≈45%. Upon further investigation, we concluded that the propagation condition associated with the outlier was too unspecific, resulting in the condition to match a large number of (irrelevant) requirements. As stated earlier, due to lack of access to the experts in Case-B, we followed a conservative approach when writing the propagation conditions, using phrases only from the changed requirement and the documented change rationale. In the case of the outlier, the rationale that was available to us was not precise enough, limiting us to use a single phrase as the propagation condition. An expert would most likely have refined the propagation condition upon seeing the sorted list; however, we did not consider such a feedback loop.

With regards to FNs, we detected *all* the impacted requirements in Case-A, and thus an FN rate of 0 for the change scenarios in this case. The domain expert involved in the case found the impact analysis results to be very useful, considering the low FP rates (Figure 6.11). For Case-B, the FN rate is 0 for all scenarios, but one (scenario B.5), where we miss 1 out of the 9 impacted requirements. The missed requirement was due to a relationship that our approach cannot identify. Specifically, the relationship is between the phrase “touristic attraction” in the missed requirement and the phrase “point of interest” in the propagation condition. The syntactic and semantic similarity measures that our approach is built upon cannot detect the tacit *is-a* relationship between the former and latter phrase. This is a limitation in our approach and needs to be addressed through further user input, e.g., a domain model.

Table 6.4. Execution Times.

Case	Task	Execution Time
<b>Case-A</b> 160 requirements 673 distinct phrases 648 distinct tokens	Phrase detection (full document)	15s
	Syntactic similarity calculation	22s
	Semantic similarity calculation	208s
	Sorted list generation (average)	13s
	Sorted list generation (worst case)	16s
<b>Case-B</b> 72 requirements 267 distinct phrases 263 distinct tokens	Phrase detection (full document)	12s
	Syntactic similarity calculation	8s
	Semantic similarity calculation	74s
	Sorted list generation (average)	9s
	Sorted list generation (worst case)	11s

**RQ4.** Table 6.4 shows the execution times for the main computational tasks in our approach. All tasks except sorted list generation are one-offs and performed in Step 1 of the process of Figure 6.2. Given the small execution times, particularly for sorted list generation, we expect our approach to scale to larger requirements documents. Execution times were measured on a laptop with a 2.3 GHz CPU and 8GB of memory.

**Threats to validity.** The lack of access to experts for specifying the change propagation conditions in Case-B poses a threat to internal validity. As discussed earlier, we mitigated this threat by having three researchers independently write the propagation conditions and limiting the choice of phrases that could be used in the conditions.

We have evaluated our approach using two case studies in different domains. The consistency of the results across these two case studies makes us optimistic about the generalizability of our approach. Nonetheless, further case studies remain essential for minimizing threats to external validity.

## 6.10 Related Work

Below, we compare our approach with related work on change impact analysis and on the application of NLP in RE.

**Change Impact Analysis (CIA).** Our work focuses on inter-requirement CIA, i.e., how requirements changes affect *other requirements*. We thus narrow our discussion to work strands that address this specific facet of CIA. Inter-requirement CIA requires some notion of dependency between requirements for change propagation. Zhang et al. [Zhang et al., 2014] study two requirements dependency models, the *D*-model [Dahlstedt and Persson, 2005] and the *P*-model [Pohl, 1996], in terms of suitability for inter-requirement CIA. They propose an enhanced model with generic dependency types such as “refines”, “conflicts”, and “constrains”. Goknil et al. [Goknil et al., 2014a] propose a similar dependency model, augmenting it with formal semantics and using it for impact analysis over NL requirements. When the requirements are expressed as models, more specialized dependency types may be used. Amyot [Amyot, 2003] uses operationalization dependencies between use cases and goals to propagate change between intentional and behavioral requirements; and Cleland-Huang et

al. [Cleland-Huang et al., 2005b] use soft goal dependencies to analyze how changes in functional requirements impact non-functional requirements.

Our work differs from the above in that it does not need the requirements dependencies to be specified ahead of time. In our context, whether there is a dependency between a changed requirement and another requirement is determined by the propagation condition. Since one cannot enumerate all possible conditions, building an explicit dependency graph is infeasible. Another difference in our approach is that it does not attempt to characterize the dependencies using a dependency model. The high expressiveness and implicit semantics of NL often make it difficult to classify dependencies using predefined types. Instead of using typed dependencies, we use similarity scores to assess the impact of changes.

A large body of work exists on automated retrieval of requirements trace links [Cleland-Huang et al., 2007, Torkar et al., 2012, Cleland-Huang et al., 2014]. Our work takes inspiration from and follows a similar process to Just-In-Time (JIT) techniques for trace retrieval [Cleland-Huang, 2012]. The main contribution of our work over existing JIT trace retrieval techniques is that we take the phrasal structure of requirements into consideration.

**NLP in RE.** NLP techniques such as tokenization, part-of-speech tagging, and similarity measurement that are used in our approach are also commonly used in, among other tasks, inconsistency and ambiguity handling, e.g., [Gervasi and Zowghi, 2005, Yang et al., 2011], requirements tracing, e.g., [Torkar et al., 2012, Cleland-Huang et al., 2014], and requirements overlap detection, e.g., [Faleski et al., 2013]. Text chunking too has been applied in RE before, albeit to a limited extent, e.g., for matching requirements to web-service descriptions [Zachos and Maiden, 2008], assessing requirements satisfaction [Holbrook et al., 2009], ambiguity resolution [Yang et al., 2011], checking conformance to templates [Arora et al., 2015a], and extracting requirements glossary terms [Arora et al., 2014a]. We use NLP to address a different problem than those addressed by the above work.

## 6.11 Conclusion

In this chapter, we presented an approach based on Natural Language Processing for analyzing the impact of changes in natural language requirements specifications. We surmised that a large majority of requirements dependencies manifest themselves within the phrasal structure of the requirements sentences. We further argued about the importance of explicitly capturing the conditions under which change should propagate between requirements. Our evaluation, over two industrial case studies, confirms our surmise, and suggests that our approach is accurate. Across the 14 change scenarios in our case studies, we could detect 99% (105 / 106) of the impacted requirements using our approach. Nevertheless, certain dependencies, an example of which was reported in our evaluation, are inherently tacit and detectable only through explicit guidance.

In the future, we plan to extend our approach with a cost-effective mechanism for explicating such dependencies through a domain model. We further plan to enhance our approach with means for handling simultaneous changes. Finally, we plan to conduct usability studies to better validate our approach. The primary focus of these future studies will be on determining whether the change propagation conditions in our approach can be elicited effectively from practitioners.



# Chapter 7

## Change Impact Analysis between SysML Models of Requirements and Design

Change impact analysis is an important activity in software maintenance and evolution, both for properly implementing a set of requested changes, and also for estimating the risks and costs associated with the change implementation [Bohner and Arnold, 1996, Pfleeger and Atlee, 2009]. In addition to being a general best practice, change impact analysis is often mandatory for safety-critical applications and meeting the compliance provisions of safety standards such as IEC 61508 [IEC, 2005] and ISO 26262 [ISO26262, 2009].

Performing this analysis manually is expensive, particularly for complex systems. In this chapter, we propose an approach to automatically identify the impact of requirements changes on system design, when the requirements and design elements are expressed using models. We ground our approach on the Systems Modeling Language (SysML) due to SysML's increasing use in industrial applications.

The approach proposed in this chapter has two steps: For a given change, we first apply a static slicing algorithm to extract an estimated set of impacted model elements. Next, we rank the elements of the resulting set according to a quantitative measure designed to predict how likely it is for each element to be impacted. The measure is computed using Natural Language Processing (NLP) applied to the textual content of the elements. Engineers can then inspect the ranked list of elements and identify those that are actually impacted. We evaluate our approach on an industrial case study with 16 real-world requirements changes. Our results suggest that, using our approach, engineers need to inspect on average only 4.8% of the entire design in order to identify the actually-impacted elements. We further show that our results consistently improve when our analysis takes into account both structural and behavioral diagrams rather than only structural ones, and the natural-language content of the diagrams in addition to only their structural and behavioral content.

**Structure.** Section 7.1 motivates the problem addressed in this chapter and provides an insight into our contributions for this chapter. Section 7.2 describes our approach. Section 7.3 presents our empirical evaluation. Section 7.4 compares with related strands of work. Section 7.5 summarizes the chapter.

## 7.1 Motivation and Contributions

In this chapter, we concern ourselves with analyzing the impact of requirements changes on system design. Changes in requirements may occur due to a variety of reasons, including, for example, evolving user needs and budget constraints. Irrespective of the cause, it is important to be able to assess how a requirements change affects the design. Doing so requires engineers to identify, for each requirements change, the system blocks and behaviors that will be impacted. If done manually, this task can be extremely laborious for complex systems, thus making it important to support the task through automation.

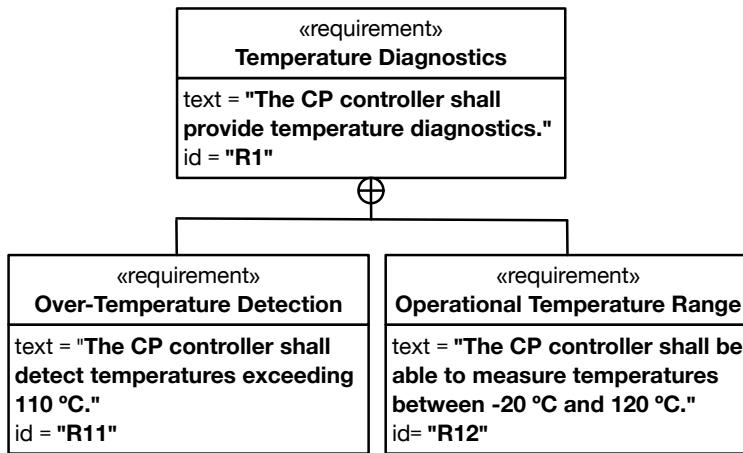


Figure 7.1. Requirements diagram fragment for CP.

**Motivating Example.** We motivate our work using a cam phaser (CP) system, developed by Delphi Automotive. This system, which includes mechanical, electronic and software components, enables adjusting the timing of cam lobes with respect to that of the crank shaft in an engine, while the engine is running. CP is safety-critical and subject to ISO 26262 – a functional safety standard for automobiles. To protect confidentiality and facilitate illustration, we have, in the description that follows, altered some of CP’s details without affecting CP’s core architecture and behavior.

The system requirements and the design of CP are expressed using the Systems Modeling Language (SysML) [INCOSE, 2016]. Figure 7.1 shows a small requirements diagram adapted from CP’s original SysML models. The requirement on the top, Temperature Diagnostics (R1), is decomposed into two sub-requirements: Over-Temperature Detection (R11) and Operational Temperature Range (R12). The over-temperature threshold specified by R11 and the operational temperature range specified by R12 depend on the specific devices that interact with CP and may vary from one engine configuration to another. Hence, it is common for systems engineers to receive change requests regarding these requirements. Examples of change requests coming from customers (typically, car manufacturers) and concerning these requirements are: (1) Ch-R11: *Over-temperature threshold shall change from 110 degC to 147 degC*, and (2) Ch-R12: *Temperature range shall be extended to -40 – 150 degC from -20 – 120 degC*.

Figures 7.2 and 7.3 present parts of CP’s design: Figure 7.2 shows a fragment of CP’s architecture expressed as a SysML internal block diagram. In this diagram, there are two traceability links to requirements, one from the Over-Temperature Monitor block (labeled  $B_2$ ) to requirement R11, and

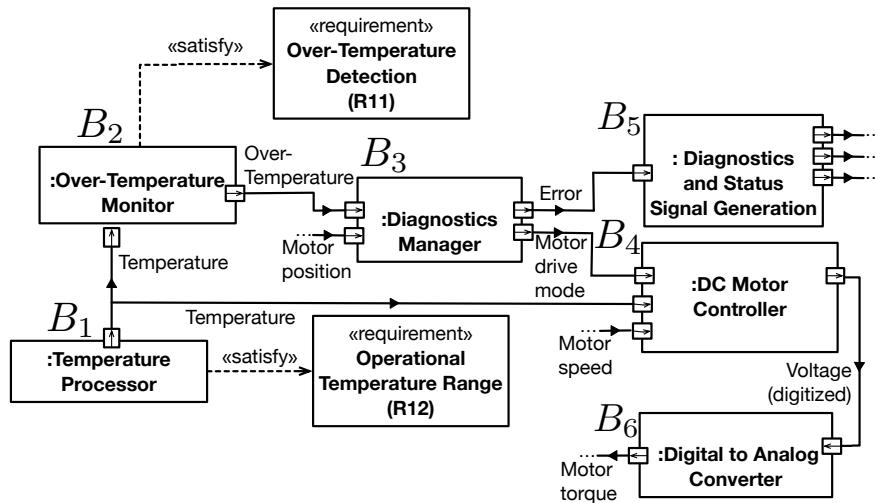
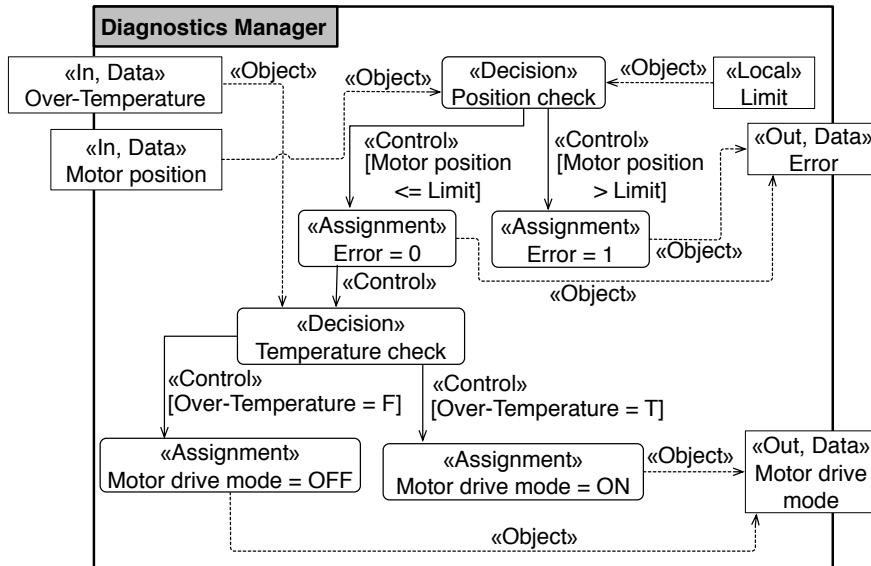


Figure 7.2. Fragment of CP's block diagram.


 Figure 7.3. (Simplified) activity diagram for the Diagnostics Manager block ( $B_3$ ) of Figure 7.2.

the other from the Temperature Processor block ( $B_1$ ) to requirement R12. Figure 7.3 shows an activity diagram describing the behavior of the Diagnostics Manager block ( $B_3$ ). For succinctness, we take the term “block” to represent *instances* of SysML block types. This choice does not cause ambiguity in our presentation, as our motivating example does not have multiple instances of the same block type.

A simple intuition that systems engineers apply for scoping the impact of requirements changes is to follow the flow of data between the design blocks, starting from the blocks that are directly traceable to the changed requirements. For example, for Ch-R11, one would start from block  $B_2$ , which is directly traced to R11, and mark as potentially-impacted any block that is reachable from  $B_2$  via the inter-block connectors. Using this kind of reasoning, we obtain the following estimated impact sets for Ch-R11 and Ch-R12, respectively:  $\{B_2, B_3, B_4, B_5, B_6\}$  and  $\{B_1, B_2, B_3, B_4, B_5, B_6\}$ .

Estimating the impact sets in the manner described above, i.e., by reachability analysis over the inter-block connectors, often yields too many false positives, i.e., too many blocks that are not actually impacted by the change under investigation. For example, we know from a manual inspection conducted by the systems engineers involved that the actual impact sets for Ch-R11 and Ch-R12 are  $\{B_2\}$  and  $\{B_1, B_4, B_6\}$ , respectively. This means that  $B_3, B_4, B_5$  and  $B_6$  in the estimated impact set for Ch-R11, and  $B_2, B_3$  and  $B_5$  in the estimated impact set for Ch-R12 are false positives.

Some of these false positives can be pruned by considering the block behaviors. To illustrate, consider the activity diagram of Figure 7.3. By following the control and data flows in this diagram, we can infer that (1) the `Motor position` input may influence both the `Error` and `Motor drive mode` outputs, and (2) the `Over-Temperature` input may influence only the `Motor drive mode` output. We can therefore conclude that  $B_5$  is unlikely to be impacted by Ch-R11 and Ch-R12, and thus remove  $B_5$  from the estimated impact sets above.

Despite the analysis of block behaviors being helpful for pruning the estimated impact sets, such analysis alone does not adequately address imprecision, still leaving the engineers with a large number of false positives and hence a large amount of wasted inspection effort. To further improve precision, we recognize that there is a wealth of textual content in the models, e.g., the labels of blocks, ports and actions. This raises the possibility that text analysis can be a useful aid for making change analysis more precise.

To this end, we use insights from our previous work on the propagation of change in natural-language content [Arora et al., 2015a]. In particular, we have observed that, alongside the change description, one can further obtain cues from the engineers about how they expect a given change to propagate. For example, the engineers of CP could provide the following intuition about the impact of Ch-R12 on the design, *before* actually inspecting the design: “Temperature lookup tables and voltage converters need to be adjusted”.

From the description of Ch-R12 and the intuition above given by the engineers, it is reasonable to expect that a block containing one or more of the keyphrases “temperature range”, “temperature lookup table”, and “voltage converter” (or similar phrases) should have a higher likelihood of being impacted by Ch-R12 than a block that contains none of these keyphrases. Indeed, the keyphrase “temperature lookup table” appears in the action nodes of the activity diagrams that describe the block behaviors of  $B_1$  and  $B_4$  (not shown), thus making  $B_1$  and  $B_4$  more likely to be impacted than other blocks, say  $B_2$  and  $B_3$ , whose activity diagrams do not contain this keyphrase. In a similar vein, the keyphrase “voltage converter” mentioned by the engineers will increase the likelihood of impact on  $B_6$  as compared to  $B_2$  and  $B_3$ .

**Contributions.** We propose an automated approach for identifying the impact of requirement changes on system design. Our approach takes into account all the intuitions illustrated on the motivating example described above, utilizing the inter-block connectors, the block behaviors, and the textual content of the models for increasing the precision of change impact analysis. Our approach has two steps: For a given change, we first compute an *estimated impact set* by identifying the design elements that are reachable from the changed requirement. The basis for this step are the inter-block connectors and block behaviors. The main novelty of this step is in providing a rigorous adaptation of dependency graphs – commonly used in program slicing [Tip, 1995] – for reachability analysis over the activity

diagrams that describe the block behaviors. In the second step, we automatically rank the elements of the estimated impact set. The ranking is aimed at predicting how likely it is for each element in this set to be affected by the given change. The basis for the ranking is a quantitative measure computed using Natural Language Processing (NLP) [Jurafsky and Martin, 2009]. Specifically, the measure reflects the similarity between the textual content of the elements in the estimated impact set and the keyphrases in the engineers' statement about the change. We provide guidelines for deciding about the cutoff point in the ranked list; this is the point beyond which the elements in the list would not be worthwhile inspecting because their likelihood of being impacted is low. The novelty of the second step of our approach is in applying NLP for change analysis between modeling artifacts (as opposed to textual artifacts).

While the ideas behind our work are general, we ground our approach on SysML. This choice is motivated primarily by two factors: First, SysML is representing a significant and increasing segment of the embedded software industry, particularly in safety-critical domains. Given the importance of change impact analysis for complying with safety standards, we believe that building on SysML is advantageous as a way to facilitate the integration of our approach into safety certification activities. Second, SysML provides a built-in mechanism, via requirements diagrams, for connecting design models to natural-language requirements. This allows us to capitalize as much as possible on the standard requirements-to-design trace link in SysML.

We implement our approach as a plugin for Enterprise Architect [EA, 2016]. We report on an industrial case study conducted in collaboration with Delphi Automotive, which is an international supplier of vehicle technology. The case study includes 16 real-world requirements changes.

Our results indicate that the number of elements engineers need to inspect decreases as we combine different sources of information. In particular, on average, this number is 21.6% of the entire design (80 / 370 design elements) when we consider only inter-block connectors. This average reduces to 9.7% (36 / 370) when we consider both inter-block connectors and block behaviors. The average further reduces to 4.8% (18 / 370) when we also take into account the natural-language information in the models and the engineers' change statements. The precision of our approach when all the above three sources of information are used is 29.4%. That is, on average, 29.4% of a set consisting of 18 elements is actually impacted. Given that the approach narrows potentially-impacted elements to a small set (4.8% of the design), excluding false positives from the results can be done without substantial effort. Our analysis misses one impacted element for only one out of the total of 16 changes. The recall is 85% for that particular change and 100% for the other 15 changes, giving an average recall of 99%.

## 7.2 Approach

Figure 7.4 shows an overview of our change impact analysis approach. In this section, we first describe the modeling prerequisites for our approach. We then elaborate the steps of our approach, marked 1 and 2 in Figure 7.4.

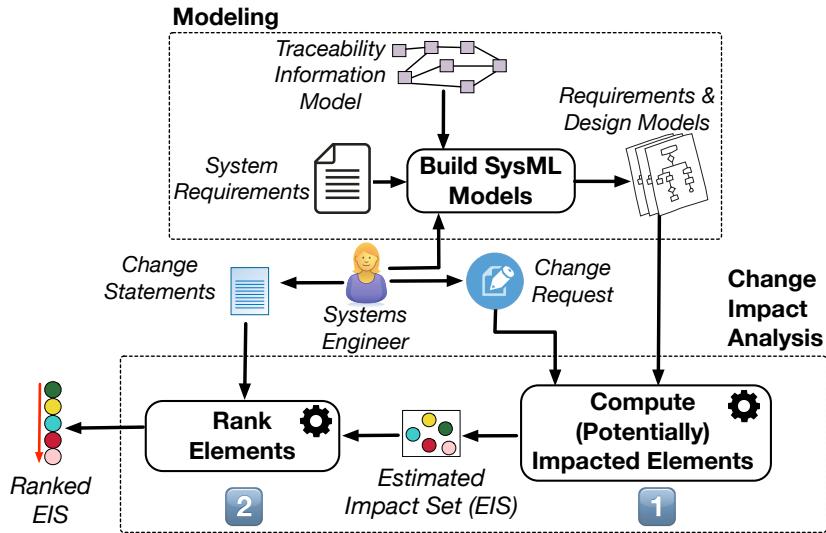


Figure 7.4. Approach overview.

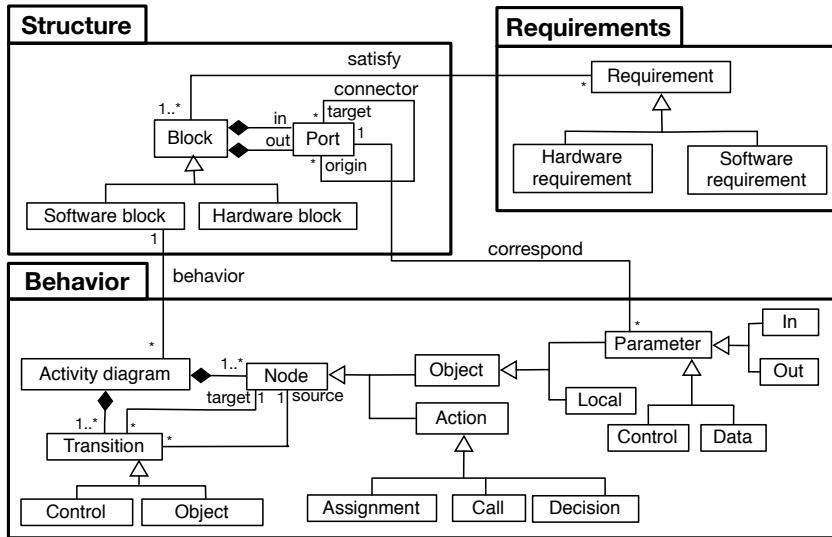
### 7.2.1 Building SysML Models

Our approach concentrates on models built along three dimensions: (1) requirements, (2) structure (architecture), and (3) behavior. We use SysML requirements diagrams, illustrated in Figure 7.1, for expressing requirements. We model the system structure using SysML internal block diagrams, illustrated in Figure 7.2. Finally, we use SysML activity diagrams, illustrated in Figure 7.3, for capturing behaviors. Our focus on these three model types is in line with SysML's core modeling practices [Friedenthal et al., 2008, Holt and Perry, 2008] and further with the modeling choices made by Delphi Automotive.

Our change impact analysis approach is agnostic to the particular modeling methodology used to build the above model types, as long as the methodology provides the traceability information required by our approach. We characterize the required traceability information via the *traceability information model (TIM)* of Figure 7.5. TIMs are a common way of specifying how different development artifacts (and the elements thereof) should be traced to one another in order to support specific analytical tasks [Cleland-Huang et al., 2014, Rempel et al., 2014, Mäder et al., 2013].

Our TIM is organized into three packages, representing the three modeling dimensions covered. This TIM is consistent with both the existing literature on systems engineering modeling [MBSE, 2008], and also the SysML/UML metamodel [Friedenthal et al., 2008, Holt and Perry, 2008]. We note that providing the information envisaged by our TIM does not require significant additional manual work. Specifically, all the elements and associations in our TIM, except for the `satisfy` association from blocks to requirements, are implied by the natural process of model construction in SysML. As for the links between requirements and system blocks prescribed by our TIM, establishing these links, irrespective of whether our change impact analysis technique is used or not, is one of the most basic best practices in SysML and, in the case of safety-critical applications, an essential prerequisite for complying with safety standards [Nejati et al., 2012, Briand et al., 2014, Sabetzadeh et al., 2011].

The requirements package in Figure 7.5 defines the requirements types, namely, `software` and



**Figure 7.5.** The traceability information required by our change impact analysis approach.

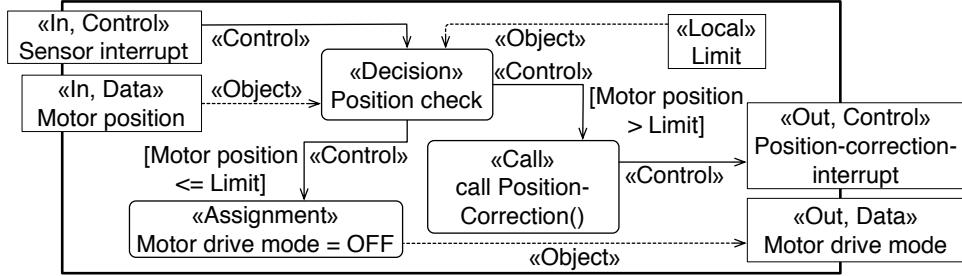
hardware. Due to lack of space, Figure 7.5 does not show all the possible relations between the requirements (e.g., decomposition and derivation). Requirements may be connected to blocks via the `satisfy` relation. A `satisfy` link between a block  $B$  and a requirement  $R$  indicates that the function implemented by  $B$  contributes to the satisfaction of  $R$ . Blocks belong to the structure package, and can be either `software` or `hardware`. Each block, irrespective of its type, contains a number of ports. Ports are connected via the `connector` relation.

Blocks can be associated with multiple behaviors (use cases). Each behavior is specified using one activity diagram. The `behavior` relation links blocks to their corresponding behaviors. Activity diagrams include nodes and transitions. Nodes can be either objects or actions; transitions may be of either the object or control kinds. Object transitions represent data flows (data dependencies), and control transitions represent control flows (control dependencies). Action nodes may be of one of the following three kinds: (1) assignment statements defined over parameters or local variables, (2) decision statements (if-statements) defined over parameters or local variables, and (3) call statements providing the behavior of function calls.

Object nodes can be designated as either parameter or local. Parameters represent the input/output variables of activity diagrams and may be of type control or data. Local nodes are used to store local variables. Control input/output variables and call action nodes are used for modeling the sending and receiving of events between blocks with concurrent behaviors. Finally, the `correspond` relation in Figure 7.5 indicates that each block port has some corresponding parameter in some activity diagram related to that block. Each activity diagram parameter has to correspond to one port of the block related to that activity diagram.

The diagrams in our motivating example of Section 7.1 conform to the TIM of Figure 7.5. Specifically, the internal block diagram of Figure 7.2 specifies the `satisfy` relations between the requirements of Figure 7.1 and blocks  $B_1$  and  $B_2$ . The activity diagram of Figure 7.3 captures one possible behavior for block  $B_3$ . This activity diagram contains two input and two output parameters, all of type data, as

well as a local variable, `Limit`. The input/output parameter nodes of the diagram correspond to the input and output ports of  $B_3$ .



**Figure 7.6.** Example activity diagram with control input/output and a call action node.

There are six action nodes in the activity diagram of Figure 7.3: four assignments and two decisions. The call action node is illustrated in the activity diagram of Figure 7.6. In this figure, the control input `Sensor interrupt` models interrupt calls that can be periodic or aperiodic. The call action node `call Position-Correction()` produces a control output value, which is in turn used as a control input by another block.

## 7.2.2 Computing Potentially Impacted Elements

In this section, we describe the first step of our change impact analysis approach (marked 1 in Figure 7.4). This step includes two algorithms: (1) computing reachability over inter-block structural relations, and (2) slicing activity diagrams based on intra-block behavioral relations.

**Reachability over inter-block structural relations.** We denote the sets of all requirements, blocks, ports and activity diagrams by  $\mathcal{R}$ ,  $\mathcal{B}$ ,  $\mathcal{P}$ , and  $\mathcal{AD}$ , respectively. Let  $r \in \mathcal{R}$ ,  $b \in \mathcal{B}$ , and  $p \in \mathcal{P}$ . We write  $satisfy(r) \subseteq \mathcal{B}$  to denote the set of blocks related to  $r$  by a `satisfy` relation. We denote the input ports of  $b$  by  $in(b) \subseteq \mathcal{P}$ , and its output ports by  $out(b)$ . We write  $connect(p) \subseteq \mathcal{P}$  to indicate the set of ports related to  $p$  by `connector` links emanating from  $p$ . Finally, we denote by  $behavior(b) \subseteq \mathcal{AD}$  the set of activity diagrams that specify the behaviors of  $b$ .

Figure 7.7 shows the algorithm for computing reachability over inter-block connectors (structural relations). The algorithm receives as input the set  $R$  of requirements modified in response to a change request. It then computes the initial sets  $iB$  of impacted blocks,  $iP$  of impacted ports, and  $iAD$  of impacted activity diagrams (lines 1–3). The initial set  $iB$  is obtained by following the `satisfy` links from the requirements in  $R$ . The set of output ports of the impacted blocks in  $iB$  are then stored in  $iP$ , and the activity diagrams related to the blocks in  $iB$  are stored in the initial set  $iAD$ .

After obtaining the initial sets, a transitive closure is computed over these sets (lines 4–9). In particular, by following the `connector` links originating from ports in  $iP$ , new input ports belonging to new blocks are identified. The new input ports are added to  $iP$  (line 5), and the new blocks are added to  $iB$  (line 6). We then identify the activity diagrams related by the `behavior` links to the newly added blocks, and add them to  $iAD$  (line 7). We further add the output ports of the newly added blocks to  $iP$  (line 9). Lines 5–8 are executed until we reach a fixed point. The result is an estimated impact set  $EIS$ . Note that for any activity diagram in  $EIS$ , we consider all the activity and object nodes in that diagram to be impacted.

---

Algorithm ComputeImpact [using *only* inter-block structural relations].

Input: - A set  $R$  of requirements modified in response to a change request.  
 - Sets  $\mathcal{B}$  (blocks),  $\mathcal{P}$  (ports) and  $\mathcal{AD}$  (activity diagrams), together with relations *satisfy*, *connect*, *in*, *out*, and *behavior*.  
 Output: - A set  $EIS$  of blocks, ports and activity diagrams impacted by  $R$ .

1.  $iB = \bigcup_{r \in R} \text{satisfy}(r)$
2.  $iP = \bigcup_{b \in iB} \text{out}(b)$
3.  $iAD = \bigcup_{b \in iB} \text{behavior}(b)$
4. **do**
5.    $iP = iP \cup \bigcup_{p \in iP} \text{connect}(p)$
6.    $iB = iB \cup \bigcup_{p \in iP} \text{in}^{-1}(p)$
7.    $iAD = iAD \cup \bigcup_{b \in iB} \text{behavior}(b)$
8.    $iP = iP \cup \bigcup_{b \in iB} \text{out}(b)$
9. **until**  $iB \cup iP \cup iAD$  reaches a fixed point
10. **return**  $EIS = iB \cup iP \cup iAD$

**Figure 7.7.** Algorithm for computing an estimated impact set ( $EIS$ ) using inter-block structural relations.

Given the example diagrams in Figures 7.1–7.3, if we execute the algorithm of Figure 7.7 for change requests Ch-R11 and Ch-R12 (i.e., by setting the input set  $R$  to  $\{R11\}$  and  $\{R12\}$  respectively), the algorithm will respectively return  $\{B_2, B_3, B_4, B_5, B_6\}$  and  $\{B_1, B_2, B_3, B_4, B_5, B_6\}$  for  $iB$ . Further, the algorithm returns the set of all the ports of the blocks in  $iB$  for  $iP$  and the activity and object nodes in Figure 7.3 for  $iAD$ .

**Slicing activity diagrams based on intra-block behavioral relations.** Our slicing algorithm operates on data and control flow dependencies among object and action nodes of activity diagrams. We first provide a formalization of activity diagrams' syntax and specify the notions of data and control dependency in our formalization. We then present our slicing technique for activity diagrams. An activity diagram  $ad$  is a tuple  $\langle AN, V, ON, G, TC, TO \rangle$  where:

- $AN$  is a set of action nodes, partitioned into three subsets  $AN^a$ ,  $AN^c$ , and  $AN^d$  representing assignment, call, and decision action nodes, respectively.
- $V$  is a set of variable names, partitioned into  $V^{io}$  and  $V^l$  indicating input/output and local variable names, respectively.
- $ON$  is a set of object nodes, partitioned into two subsets  $ON^p$  and  $ON^l$  indicating parameter and local object nodes, respectively. The set  $ON^p$  of parameter nodes is partitioned into  $ON^{p,c}$  and  $ON^{p,d}$  indicating call and data object nodes, respectively. The set  $ON^p$  is also partitioned into  $ON^{p,i}$  and  $ON^{p,o}$  denoting input and output object nodes, respectively. We thus have:  $ON^p = ON^{p,c} \cup ON^{p,d} = ON^{p,c} \cup ON^{p,d}$ . Each object node in  $ON^l$  (resp.  $ON^p$ ) is labeled with a variable name in  $V^l$  (resp.  $V^{io}$ ).
- $G$  is a set of Boolean expressions over the variables in  $V$ . These expressions capture transition guards.
- $TC$  is a set of control transitions defined as follows:  $TC \subseteq ((ON^{p,c} \cap ON^{p,i}) \times AN) \cup (AN^c \times (ON^{p,c} \cap ON^{p,o})) \cup ((AN^a \cup AN^c) \times AN) \cup (AN^d \times G \times AN)$ . That is, control transitions connect (1) input control parameter nodes to action nodes, (2) call action nodes to output control parameter nodes,

Algorithm ComputeImpact [using both inter-block structural and intra-block behavioral relations].

Input: - A set  $R$  of requirements modified in response to a change request.  
 - Sets  $\mathcal{B}$  (blocks),  $\mathcal{P}$  (ports), and  $\mathcal{AD}$  (activity diagrams) together with the relations *satisfy*, *connect*, *in*, *out*, and *behavior*.

Output: - A set  $EIS$  of blocks, ports and activity diagrams impacted by  $R$ .

```

1.  $iB = \bigcup_{r \in R} \text{satisfy}(r)$ 
2.  $iP = \bigcup_{b \in iB} \text{out}(b)$ 
3.  $iAD = \bigcup_{b \in iB} \text{behavior}(b)$ 
4. do
5.    $iP = iP \cup \bigcup_{p \in iP} \text{connect}(p)$ 
6.   for  $b \in \bigcup_{p \in iP} \text{in}^{-1}(p)$  do
7.      $iB = iB \cup \{b\}$ 
8.     for  $ad \in \text{behavior}(b)$  do
9.        $iAD', iOut = \text{FORWARDSLICE}(ad, \text{correspond}^{-1}(iP \cap \text{in}(b)))$ 
10.       $iAD = iAD \cup iAD'$ 
11.       $iP = iP \cup \text{correspond}(iOut)$ 
12.   until  $iB \cup iP \cup iAD$  reaches a fixed point
13. return  $EIS = iB \cup iP \cup iAD$ 

```

**Figure 7.8.** Algorithm for computing  $EIS$  using both structural and behavioral relations.

(3) assignment and call action nodes to action nodes, and (4) decision action nodes to action nodes. The transitions between decision action nodes and action nodes (the fourth item) are guarded by Boolean expressions in  $G$ .

-  $TO$  is a set of object transitions, defined as follows:  $TO \subseteq (ON^l \times (AN^d \cup AN^a)) \cup (AN^a \times ON^l) \cup ((ON^{p,d} \cap ON^{p,i}) \times (AN^d \cup AN^a)) \cup (AN^a \times (ON^{p,d} \cap ON^{p,o}))$ . That is, object transitions connect (1) local object nodes to decision and assignment action nodes, (2) assignment action nodes to local object nodes, (3) input data parameter nodes to decision and assignment action nodes, and (4) assignment action nodes to output data parameter nodes.

In our formalization of activity diagrams, we have excluded pseudo-nodes, namely the initial, final, fork, join, and merge nodes. Since the activity nodes in our formalization can have multiple incoming and outgoing transitions, pseudo-nodes do not lead to additional semantics. The semantics of our activity diagrams is identical to that described in [Bock, 2006]. Action nodes are the basic building blocks receiving inputs and producing outputs, called *tokens*. Tokens correspond to anything that flows through transitions. Tokens can be either data or control. Data tokens transit through object transitions ( $TO$ ) and carry (partial) result values. Control tokens, however, transit through control transitions ( $TC$ ) and carry null values.

Control tokens are meant to be used as triggers (events). Data tokens are associated with data parameter nodes and local object nodes, while control tokens are associated with control parameter nodes. An action node can start its execution only when *all* its input tokens via its incoming control or object transitions are provided. Upon its completion, an action node produces appropriate data and control tokens on its outgoing control and object transitions.

Algorithm. ForwardSlice.

Input: An activity diagram  $ad = \langle AN, V, ON, G, TC, TO \rangle$ .  
 A set  $in$  of *impacted* input parameter nodes of  $ad$  (slicing criterion).  
 Output: A set  $iOut$  of impacted output parameter nodes of  $ad$ .  
 A set  $iAD$  of impacted object and action nodes of  $ad$ .

```

1.  $iOut = \emptyset; iAD = \emptyset;$ 
2.  $R = TC \cup TO$ 
2. for  $n \in in \cap ON^{p,i}$  do
3.    $X = \{n\}$ 
4.   do
5.      $X = X \cup \{n' \mid \exists q \in X \cdot R(q, n') \vee \exists g \in G \cdot R(q, g, n')\}$ 
6.   until  $X$  reaches a fixed point
7.    $iAD = iAD \cup X$ 
8.    $iOut = iOut \cup (X \cap ON^{p,o})$ 
9. return  $iN, iOut$ 

```

**Figure 7.9.** Algorithm for activity diagram slicing (used by the algorithm of Figure 7.8).

Assignment action nodes receive both control and data tokens as input and generate both data and control tokens as output (e.g., see Figure 7.3). Values are passed from one assignment action node to another via local object or parameter nodes. In other words, an assignment action node sends its output to a local object or a parameter node that is connected to the input of another action node. Call action nodes receive control tokens as input and produce control tokens as output (e.g., see Figure 7.6). Function calls are modeled by a call action node generating a token on an output control parameter node of the caller that is linked to some input control parameter node of the callee. Decision action nodes receive both control and data tokens as input, but generate only control tokens as output (e.g., see Figure 7.3).

Having described our formalization of activity diagrams, we now explain, using this formalization, how we account for block behaviors in the computation of estimated impact sets. The drawback of the algorithm presented earlier in Figure 7.7 is that it naively identifies all the output ports of any impacted block as impacted without regard to the intra-block dependencies between the impacted input ports and the output ports of that block. To address this drawback, we improve our algorithm as shown in Figure 7.8.

The modified algorithm of Figure 7.8 forward slices the activity diagrams related to the impacted blocks, starting from their impacted input ports. Recall from Figure 7.5 that each activity parameter node corresponds to some input port of a block related to that activity diagram. Our slicing criterion (i.e., the starting point) is described in terms of impacted input parameter nodes. In Figure 7.8, we use  $correspond(n)$  to denote the block ports related to an activity parameter node  $n$ , and use  $correspond^{-1}(iP)$  to denote the set of activity parameter nodes related to the ports in  $iP$ .

Our forward static slicing algorithm, shown in Figure 7.9, is similar to existing forward program slicing approaches where slices are computed over *program dependency graphs* [Tip, 1995]. In these graphs, nodes correspond to program statements and are connected by edges representing control and data dependencies. The object transitions  $TO$  in our activity diagrams correspond to program def-use chains, specifying data dependencies [Alomari et al., 2012]. The control transitions  $TC$  capture control dependencies from decision nodes to sequences of action nodes in the if-then-else branches

as well as control dependencies between call action nodes and input/output control parameter nodes. The algorithm in Figure 7.9 computes, for any input parameter node  $n$  in the slicing criterion set ( $in$ ), all the action and object nodes reachable from  $n$  via sequences of object and control transitions ( $TC \cup TO$ ). The algorithm then returns all the reachable nodes ( $iAD$ ) and all the reachable output parameter nodes ( $iOut$ ).

For example, suppose that the slicing algorithm is called with  $ad$  set to the activity diagram of Figure 7.3, and  $in$  set to the `Over-Temperature` input parameter node. The algorithm computes for this activity diagram a forward slice such that: (1) the set  $iAD$  contains the decision node `Temperature check`, the assignment nodes `Motor drive mode = OFF` and `Motor drive mode=ON`, and the `Motor drive mode` output parameter node. And, (2) the set  $iOut$  contains the `Motor drive mode` output parameter node which corresponds to an output port of block  $B_3$  with the same label. Hence, the `Motor drive mode` output port of  $B_3$  is the only output port that is likely to be impacted if a change is made to the `Over-Temperature` input port of  $B_3$ . Therefore, using the modified algorithm of Figure 7.8, we prune  $B_5$ , its ports, and its related behaviors from the *EISs* computed for change requests Ch-R11 and Ch-R12.

An important remark about the computation of *EISs* in our approach is that this process is meant to be *intertwined* with the implementation of a given change request. This intertwining provides a human feedback loop where the changes made to the models by the engineers at any given step is used for improving the accuracy of the *EIS* computed in the next steps. In particular, if the engineers modify the inter-block or intra-block dependencies in the SysML models during the implementation of a change request, the *EIS* needs to be recomputed. If no changes are made to the inter-block or intra-block paths, e.g., as is the case for Ch-R11 and Ch-R12 in our motivating example, the *EIS* will remain unaffected.

### 7.2.3 Ranking Potentially Impacted Elements

In the second step of our approach (marked 2 in Figure 7.4), we rank the elements of the *EIS* computed by the previous step. In addition to the *EIS*, this second step requires one or more natural-language statements from the engineers. These statements, which we call *change statements*, include the change description as well as any intuition that the engineers may have, based on their domain knowledge, about how a certain requirements change would propagate to the design. We denote the set of change statements by  $chStat$ .

Our ranking of the elements in the *EIS* is based on matching the natural-language labels of these elements against the keyphrases that appear in the statements of  $chStat$ . The keyphrases are extracted automatically using a keyphrase extractor. We use a tailored extractor that we developed in our previous work [Arora et al., 2015b] for supporting requirements tasks. For example, applying the extractor over the statement “Temperature lookup tables and voltage converters need to be adjusted.” given by the engineers for Ch-R11 (as discussed in Section 7.1) would identify the following keyphrases: “temperature lookup table” and “voltage converter”.

With the keyphrases extracted, we use *similarity measures* to quantify how closely the text labels of the *EIS* elements match the keyphrases. Similarity measures can be *syntactic* or *semantic*. Syntactic measures are based on the string content of text segments (sometimes combined with frequencies). An example syntactic measure is *Levenshtein* [Manning et al., 2008], which computes a similarity score

between two strings based on the minimum number of character edits required to transform one string into the other. Semantic measures are calculated based on relations between the meanings of words. An example semantic measure is *Path* [Rus et al., 2013], which computes a similarity score between two words based on the shortest path between them in an is-a hierarchy (e.g., an “automobile” is-a “vehicle” and so is a “train”).

Similarity measures, both syntactic and semantic, are typically normalized to a value between 0 and 1, with 0 signifying no similarity and 1 signifying a perfect match. In line with common practice [Rus et al., 2013], we zero-out similarity scores below a certain threshold, in this chapter 0.05, to minimize noise. In addition to individual similarity measures, we further consider pairwise combinations of syntactic and semantic measures due to these measures having a complimentary nature [Nejati et al., 2012]. For the combination, we take the maximum of the two computed scores. Since there are several similarity measures to choose from, it is important to empirically investigate which measures are most suited to a specific task. Finding the best measures for our application context is addressed in our empirical evaluation (see Section 7.3).

Given an individual or a combined similarity measure, we compute for every  $e \in EIS$  the similarity between the text label of  $e$  and the keyphrases obtained from *chStat*. The score we assign to  $e$  is the largest similarity score between  $e$  and any of the keyphrases. We then sort the elements of *EIS* in descending order of the scores assigned to the elements. The assumption here is that these scores are correlated with the likelihood of the elements being actually impacted by the change under analysis. In other words, we take the elements ranked higher in the sorted *EIS* to be more likely to be impacted. For example, for Ch-R12, we would obtain high scores for any block, port, activity node, or activity transition in the *EIS* that has a high degree of similarity to either “temperature lookup table” or “voltage converter”.

## 7.3 Empirical Evaluation

In this section, we investigate through an industrial case study the following Research Questions (RQs):

**RQ1. (Usefulness of Slicing)** *How much reduction in the size of EIS does our slicing technique bring about? Does slicing remove any actually-impacted elements (true positives) from the EIS computed by inter-block structural analysis?* With RQ1, we study the usefulness of our behavioral analysis by comparing the EISs obtained from structural analysis only (i.e., the algorithm of Figure 7.7) versus those obtained from both structural and behavioral analysis (i.e., the algorithm of Figure 7.8). In particular, we are interested in the magnitude of reductions in the EIS size that our behavioral analysis provides without compromising the recall, i.e., without removing actually-impacted elements from the EIS.

**RQ2. (Choice of Similarity Measures)** *Which similarity measures are best suited to our approach?* There are several syntactic and semantic similarity measures for textual content. The choice of sim-

ilarity measures used for calculating impact rankings directly affects the quality of our results. With RQ2, we identify the syntactic and semantic similarity measures that lead to the most accurate results.

**RQ3. (Usability)** *How should engineers use the ranked EIS lists produced by our approach?* For our approach to be useful, engineers need to determine how much of a ranked EIS list is worth inspecting. In other words, they need to determine a point in the list beyond which the remainder of the list is unlikely to contain impacted elements. With RQ3, we aim to develop systematic guidelines for inspecting the ranked EIS lists.

**RQ4. (Effectiveness)** *How effective is our automated approach when compared to a manual analysis performed by an engineer?* Assuming that the guidelines resulting from RQ3 are followed, RQ4 aims to determine whether our approach can reliably identify the set of actually impacted elements, and at the same time, save substantial inspection effort.

**RQ5. (Scalability)** *Does our approach have an acceptable execution time?* With RQ5, we study whether the execution time of our approach is practical.

**Industrial Subject.** Our case study is the cam phaser (CP) system introduced in Section 7.1. A SysML model for CP had been developed by the Delphi engineers in the Enterprise Architect tool [EA, 2016]. This case study model consists of seven requirements diagrams containing 34 requirements, nine internal block diagrams with 48 blocks, 19 activity diagrams, and 56 traceability links, all of type *satisfy*. In total, the entire CP model contains 370 blocks, ports, and activity and object nodes. We chose CP as our case study model since it is an industrial system. The SysML model of CP contains a reasonable number of requirements and traceability links from requirements to blocks.

We were provided with 16 requirements change scenarios for CP. These scenarios are *real* and drawn from change requests originating from the customers of CP. In each case, a high-level natural-language statement was available which described the change as well as how the engineers expected the change to affect the design. One of these statements, referred to in our approach as a change statement, was illustrated in the motivating example of Section 7.1 (for Ch-R12). Five additional examples of change statements are provided in Table 7.1. As seen from the table, the statements are abstract and do not exactly pinpoint the impact of a change. Nevertheless, the keyphrases in the statements provide a mechanism for ranking the estimated impact sets computed by our approach. The actual impact set for each of the 16 change scenarios was further provided by the engineers involved in the case study.

**Implementation.** We have implemented our approach as a plugin for the Enterprise Architect modeling environment [EA, 2016]. Our implementation enables users to automatically generate EISs using the algorithms of Section 7.2.2, and compute ranked EISs based on NLP similarity measures as discussed in Section 7.2.3. Our plugin is available at:

[https://bitbucket.org/carora03/cia\\_addin](https://bitbucket.org/carora03/cia_addin)

**Metrics.** We use two well-known metrics, *precision* and *recall*, in our evaluation. Precision measures quality (i.e., low number of false positives) and is the ratio of actually-impacted elements found in an EIS to the size of the EIS. Recall measures coverage (i.e., low number of false negatives) and is

**Table 7.1.** Additional examples of change statements

id	Change Statements
1	Resolution of the battery voltage shall change from 0.1v to 0.01v. Battery voltage variables should be checked.
2	Input voltage divider of the battery voltage reading shall change from 0.2v to 0.3v. Measurement routines should be adjusted.
3	The motor control routine shall be executed in the 1ms task instead of the 2ms task. Slow regulator tasks should be revised.
4	Motor current shall increase from 45A to 50A. Shunt values and resolution of variables measuring the current should be revised.
5	The current mirror for the motor current measurement shall be replaced by a differential amplifier. Resolution settings should be revised.

the ratio of the actually-impacted elements found in an EIS to the number of all actually-impacted elements.

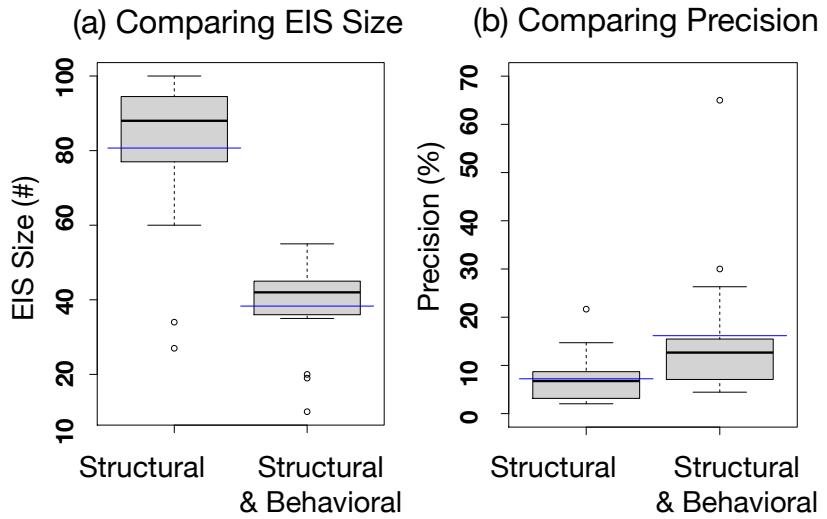
**Results.** Next, we discuss our RQs:

**RQ1. (Usefulness of Behavioral Analysis)** To answer this RQ, we compare the EISs generated by the two algorithms in Figures 7.7 and 7.8, using the CP SysML model and the 16 given change scenarios. Recall that the algorithm in Figure 7.7 relies on structural inter-block relations only, and the one in Figure 7.8 uses both structural inter-block and behavioral intra-block relations. We obtained 16 EISs via structural analysis and 16 other EISs via combined structural and behavioral analysis. We compared the sizes of the EISs, and their recall and precision values. The recall for all the 16 changes and for both the structural and the combined structural and behavioral approaches were 100%.

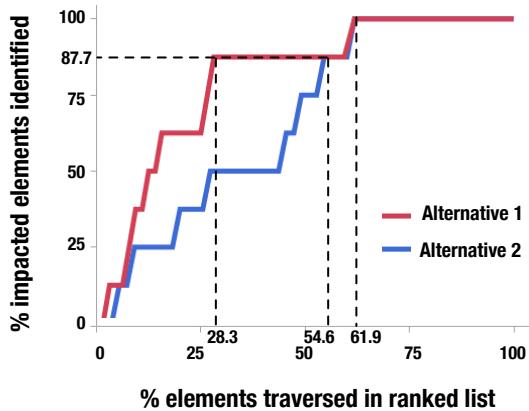
Figure 7.10 compares the EIS size and the EIS precision distributions for the 16 changes obtained by structural analysis and by the combined structural and behavioral analysis. The average EIS size and EIS precision based on structural analysis are, respectively, 80 and 8%, and based on the combined analysis are 38 and 16%, respectively. That is, after applying the forward slicing used in our behavioral analysis, on average, the EIS size is reduced by around 42 elements and the precision increases by 8%. We note that the total number of elements in the entire SysML model is 370. Hence, the EIS size generated by the structural analysis contains 21.6%, and the EIS size obtained by the combined analysis contains 9.7% of all the design elements. *In summary*, our results show that applying the combined structural and behavioral analysis significantly reduces the EIS size and increases precision without a negative effect on recall.

**RQ2. (Choice of Similarity Measures for Ranking)** To answer this RQ, we define a notion of accuracy for the ranked EISs (obtained from the ranking step in Section 7.2.3). We conceptualize accuracy using charts that show the percentage of actually-impacted elements identified (Y-axis) against the percentage of EIS elements traversed in the ranked list (X-axis). Figure 7.11 shows charts for an EIS, computed for one of the CP change scenarios, and ranked by two alternative applications of similarity measures.

A simple way to compare the alternatives would be the following: A similarity measure *A* (potentially, the combination of a syntactic and a semantic measure) is better than a measure *B* if the engineers are able to identify all the impacted elements by inspecting fewer elements when they use



**Figure 7.10.** Impact of behavioral analysis: Comparing (a) the size of EISs and (b) the precision of EISs obtained by structural analysis alone and by combined behavioral and structural analysis.



**Figure 7.11.** Two alternative applications of similarity measures for ranking the same EIS.

the list ranked by *A* than when they use the list ranked by *B*. This intuition however, cannot distinguish the two alternatives in Figure 7.11, as both identify all the actually-impacted elements after traversing 61.9% of the elements.

Despite the above, Alternative 1 is better than Alternative 2 because it produces better results *earlier*. Specifically, if the engineers choose to stop, say after inspecting 30% of the list, with Alternative 1, they will find 87.7% of the actually-impacted elements. Identifying the same percentage of actually-impacted elements with Alternative 2 would require the inspection of  $\approx 55\%$  of the list. To reward earlier identification of impacted elements, we use the Area Under the Curve (AUC) for evaluating accuracy. AUC can tell apart the two alternatives in Figure 7.11, as the metric is larger for Alternative 1.

We considered pairwise combinations of three syntactic measures, SoftTFIDF, Monge\_Elkan and

Levenshtein, from the SimPack library [SimPack, 2016] and four semantic measures, LIN, PATH, RES and JCN, from the SEMILAR library [Rus et al., 2013]. We further considered alternatives where only a syntactic or only a semantic measure is applied. Specifically, we considered 19, i.e.,  $(4+1) \times (3+1) - 1$ , alternatives. We ran these alternatives on our 16 changes, and obtained an AUC for each alternative.

To identify the best alternative for similarity measures, we need to determine which alternatives consistently result in the highest accuracy (i.e., AUC) when applied to the change scenarios. This analysis is often done using a regression tree [Breiman et al., 1984], which is a hierarchical partitioning of a set of data points aimed at minimizing, with respect to a given metric, variations across partitions. In our context, each data point is a similarity measure alternative applied to an individual change scenario. We therefore have a total of  $19 * 16 = 304$  data points. The metric of interest here is AUC.

In Figure 7.12, we show the regression tree for our case study. In each node of the tree, we show the count (number of AUC values), the mean and standard deviation for the AUC values, the similarity measures generating the AUC values in that partition, and, for every non-leaf node, the difference between the mean AUC values of its right and left children. At each level, the factor (either syntactic measure or semantic measure) that best explains the variation in AUC values is selected. The partitioning at the first level of the tree signifies the most influential factor explaining the variation observed across the data points. In Figure 7.12, the most influential factor is the choice of syntactic measure.

The nodes on the right are of particular interest, as they signify the alternatives that result in higher AUC values on average than the alternatives of the left node. We iteratively partition the right-most nodes until the difference between the mean AUC values in the resulting branches is insignificant. We deem differences below 0.01 to be insignificant. At the first level, NONE and SoftTFIDF perform better than Monge\_Elkhan and Levenshtein. Depending on whether NONE appears in a syntactic or a semantic measure node, it indicates the stand-alone application of measures of the other type. For example, the NONE appearing alongside SoftTFIDF at the first level indicates stand-alone application of semantic measures. At the second level, the syntactic alternatives are further split suggesting that SoftTFIDF produces better results on average than NONE (i.e., stand-alone application of any semantic measure). The right-most node at the last level of the tree contains the most robust alternatives that yield the highest AUC values. The reason why SoftTFIDF performs best is because it filters noise: the measure assigns a zero (or very low) score to phrase pairs when their constituent tokens are not closely matching (lower than a certain threshold), or when the matching tokens are very common (e.g., stopwords).

*In summary*, and as suggested by the right-most leaf node in the tree of Figure 7.12, SoftTFIDF (syntactic measure) combined with either RES or JCN (semantic measures) would be the most suitable alternative. We use these two alternatives to answer RQ3 and RQ4.

**RQ3. (Usability)** To answer RQ3, we aim to find the best trade-off between the number of elements to inspect and the number of impacted elements found. We define a notion of *cutoff* indicating the percentage of a ranked list which is worthwhile inspecting, and hence, should be recommended to engineers. To define a cutoff, we use delta charts, as illustrated in Figure 7.13 for one of the change scenarios in CP. In the chart, at any position  $i$  on the  $X$ -axis, the  $Y$ -axis is the difference between the

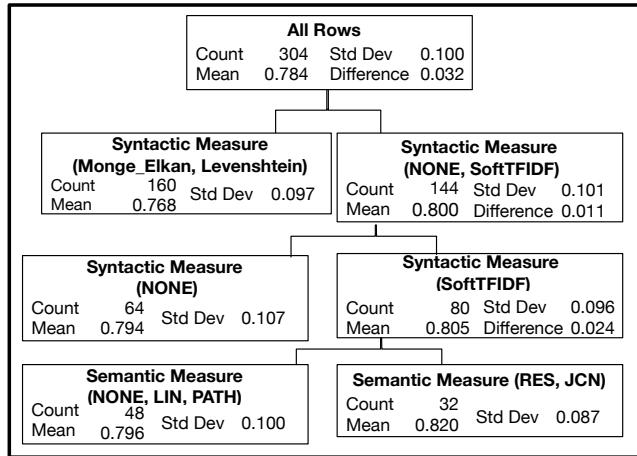


Figure 7.12. Regression tree for identifying the best similarity measure alternatives.

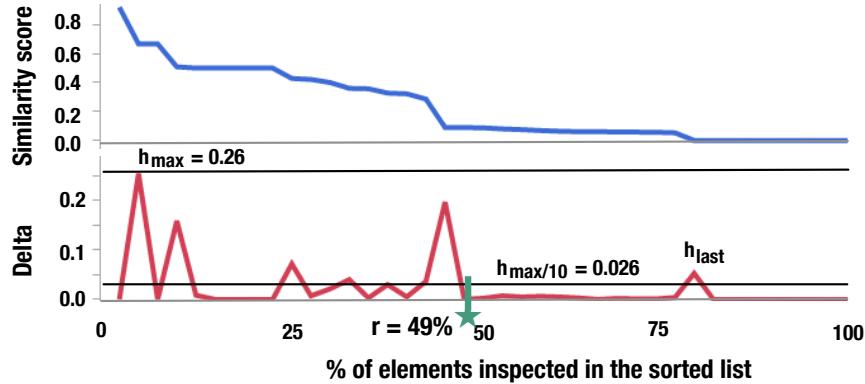
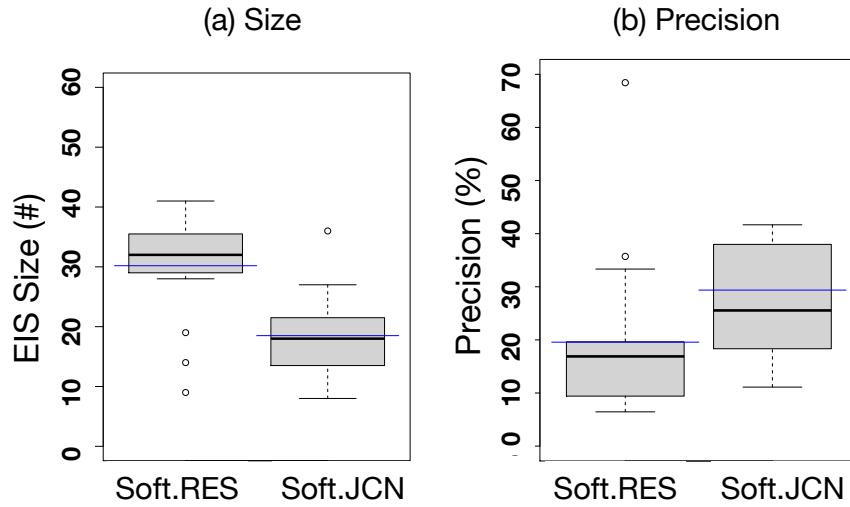


Figure 7.13. Ranked similarity scores and delta chart for an example change scenario from CP. The delta chart is used for computing the cutoff ( $r$ ).

similarity scores at positions  $i$  and  $i - 1$ . For easier understanding, in Figure 7.13, we further show the ranked similarity scores on the top of the delta chart. These similarity scores were computed using SoftTFIDF (syntactic measure) and JCN (semantic measure). As described in Section 7.2.3, the label of each EIS element  $e$  is compared against all keyphrases in the change statement using both SoftTFIDF and JCN. The maximum value obtained from all these comparisons is assigned to  $e$  as its similarity score. The chart on the top of Figure 7.13 plots the EIS elements in descending order of the similarity scores.

For the cutoff, we pick the point on the X-axis after which there are no significant peaks in the delta chart. Intuitively, the cutoff is the point beyond which the similarity scores can no longer adequately tell apart the elements in terms of being impacted. What is a significant peak is relative. Based on our experiments, a peak is significant if it is larger than one-tenth of the highest peak in the delta chart, denoted  $h_{max}$  in Figure 7.13. The only exception is the peak caused by zeroing out similarity scores smaller than 0.05 (see Section 7.2.3). This peak, if it exists, is always the last one and hence denoted  $h_{last}$ . Since  $h_{last}$  is artificial in the sense that it is caused by zeroing out negligible similarity values, we ignore  $h_{last}$  when deciding about the cutoff.



**Figure 7.14.** Size and precision of EISs that result from the application of the guidelines of RQ3 to the EISs computed by the algorithm of Figure 7.8.

More precisely, we define the cutoff  $r$  to be at the end of the right slope of the last significant peak (excluding  $h_{last}$ ). In the example of Figure 7.13,  $h_{max} = 0.26$ . Hence,  $r$  is at the end of the last peak with a height  $> h_{max}/10 = 0.026$ . We recommend that engineers should inspect the EIS elements up to the cutoff and no further. In the example of Figure 7.13, the cutoff is at 49% of the ranked list. We note that the cutoff can be computed *automatically* and without user involvement. Therefore, the delta charts and their interpretation are transparent to the users of our approach.

*In summary*, for each change scenario, we automatically recommend, through the analysis of the corresponding delta chart as explained above, the fraction of the ranked EIS that the engineers should manually inspect for identifying actually-impacted elements.

**RQ4. (Effectiveness)** To answer RQ4, we report the results of applying the best similarity measure alternatives from RQ2 for ranking the EISs computed by the algorithm of Figure 7.8 (i.e., combined structural and behavioral analysis), and then considering only the ranked EIS fractions recommended by the guidelines of RQ3. Note that in this RQ, by EIS we mean the fraction obtained after applying the guidelines of RQ3. In Figure 7.14, we show for our 16 changes the size and precision distributions of the recommended EISs. These distributions are provided separately for the best similarity alternatives from RQ2, i.e., SoftTFIDF combined with RES (denoted Soft.RES) and SoftTFIDF combined with JCN (denoted Soft.JCN).

The average EIS size is 30.2 for Soft.RES and 18.5 for Soft.JCN. The average precision for Soft.RES and Soft.JCN are 19.5% and 29.4% respectively. As for recall, Soft.RES yields a recall of 100% for all 16 changes, while Soft.JCN misses one element for one change. That is, using Soft.JCN, we have a recall of 100% for 15 changes, and a recall of 85% for one change (i.e., an average recall of 99%). The results clearly show that Soft.JCN yields better overall accuracy.

*In summary*, after applying our best NLP-based similarity measure, Soft.JCN, the average precision of our analysis increases to 29.4% compared to 16% obtained by the combined behavioral and structural analysis (discussed in RQ1). The average recall reduces to 99% compared to 100% ob-

tained by the combined analysis. Finally, using NLP, the average number of elements to be inspected by the engineers reduces to 18.5 (just 4.8% of the entire design model) compared to 38 (9.7% of the design model) before applying NLP.

**RQ5. (Execution Time)** The execution time for both steps of our approach, i.e., computing the EISs and ranking the EISs, was in the order of seconds for the 16 changes. Given the small execution times, we expect our approach to scale to larger systems. Execution times were measured on a laptop with a 2.3 GHz CPU and 8GB of memory.

**Validity considerations and threats.** Internal and external validity are the most relevant dimensions of validity for our case study. With regard to internal validity, an important consideration is that the change statements must represent the understanding of the engineers about a change *before* the engineers have determined the impact of that change; otherwise, the engineers may learn from the analysis they have performed and provide more precise change statements than when they have not examined the design yet. If this occurs, the accuracy results would not faithfully represent what one can achieve in a non-evaluation setting. In our case study, the change statements were pre-existing and written at the time that the change requests had been filed, i.e., before the impact of the changes had been examined. The engineers in our case study were therefore required only to inspect the design and provide the actual impact sets (gold standard). Consequently, learning is not a significant threat to internal validity. With regard to external validity, while our case study is industrial and we anticipate it to be representative of many embedded systems, particularly in safety-critical domains, additional case studies will be essential in the future.

## 7.4 Related Work

There is a wide range of techniques for change impact analysis covering various development artifacts, including requirements, design and code [Lehnert, 2011]. Our work is concerned with analyzing how changes made to *requirements* will impact *system design*, in a context where the requirements are expressed using natural language, and the design using models. This situation is common and particularly relevant for embedded systems development. Below, we compare with the existing work that is most pertinent to the context in which we studied change impact analysis in this chapter.

Any change impact analysis technique for requirements and design artifacts has to account for the dependencies between requirements and design elements to properly propagate changes. Aryani et al. [Aryani et al., 2010] use logical relationships between domain concepts described in terms of weighted dependency graphs. van den Berg [van den Berg, 2006] augments traceability links with dependency type information between software artifacts. Goknil et al. [Goknil et al., 2014a] extend the approach of van den Berg [van den Berg, 2006] with formal semantics and apply it for impact analysis over requirements. When the requirements are expressed as models, more specialized dependency types may be defined. For example, Cleland-Huang et al. [Cleland-Huang et al., 2005a] use soft goal dependencies to analyze how changes in functional requirements propagate to non-functional requirements. Tang et. al. [Tang et al., 2007] capture dependencies using a special model, called an architectural rationale and linkage model, and use this model alongside probabilistic expert estimates for change impact analysis over system architectures.

Our work differs from the above in that we do not require dependency types, logical relationships

between domain elements, architectural rationale, or probabilistic data for inferring or estimating impact likelihoods. Instead, we utilize the textual content of the design models and simple natural-language statements from engineers for impact likelihood prediction. The high expressiveness and implicit semantics of textual data make it difficult to come up with a crisp classification of dependencies or to obtain precise logical relations between the requirements and design. This is why we use (quantitative) similarity measures for predicting impact likelihoods.

A number of approaches rely on pre-defined rules for change propagation. Briand et al. [Briand et al., 2003] propose a taxonomy of model changes based on the UML metamodel, and use this taxonomy in order to specify rules for identifying which parts of a UML model need to be updated after each change so that the model will remain consistent with the UML metamodel. Müller and Rumpe [Müller and Rumpe, 2014] propose a domain-specific language for the specification of impact rules. These rules capture what kind of changes to models lead to what kind of impact. The main goal of these rule-based approaches is to maintain the consistency of models after changes; these approaches are not targeted at analyzing the impact of requirements changes. Our work has a different focus, as it aims at analyzing the impact of requirements changes on design. We do not use pre-defined impact rules in our approach.

Furthermore, there are approaches that rely on historical information obtained from software repositories for change propagation. For example, Wong and Cai [Wong and Cai, 2009] combine logical relationships between UML class diagrams and historical information obtained from software repositories to predict the scope of the impact of a given change. This approach relies on the existence of complete and consistent version histories. Such versioning is typically used for keeping track of changes in implementation-level artifacts, e.g., code. Our approach does not rely on historical data and can be used effectively in situations where no historical data is available, notably in early stages of development, or where detailed versioning of models is not practiced.

Control flow analysis techniques (e.g., software method call dependencies) have been used before to automatically identify traceability links [Dit et al., 2013], and further to facilitate code-level change impact analysis [Li et al., 2013]. Kuang et al. [Kuang et al., 2012] demonstrate that combining control flow and data dependencies improves automated retrieval of traceability links from requirements to code. Our results in this chapter lead to a similar conclusion in the context of model-based development, that is, leveraging both inter-block data flow dependencies and intra-block control and data flow dependencies improves the accuracy of change impact analysis.

In our previous work [Arora et al., 2015b], we already used similarity measures as change impact predictors. Nevertheless, this earlier work was focused on inter-requirement change impact analysis, i.e., identifying the impact of requirements changes on other *requirements*, rather than on the design. This earlier work focused exclusively on natural-language content. In our current work, we generalize our previous work to a model-based development setting, where we exploit not only the natural-language content of the development artifacts but also the structure and semantics of the design models.

## 7.5 Conclusion

In this chapter, we presented an approach to automatically identify the impact of requirements changes on system design. Our approach has two main steps: First, for a given change, we obtain a set of estimated impacted model elements by computing reachability over inter-block data flow and intra-block control and data flow dependencies. Next, we rank the resulting set of elements according to a quantitative measure obtained using NLP techniques. The measure reflects the similarity between the textual content of the elements in the estimated impact set and the keyphrases in the engineers' statements about the change.

Our evaluation on an industrial system shows that the accuracy of our approach consistently improves when we consider both inter-block and intra-block dependencies rather than only the inter-block ones, and the textual content of the diagrams in addition to only the elements' dependencies. Although not included in this chapter, we note that eliminating the analysis in the first step of our approach and only applying the NLP technique in the second step reduces the accuracy considerably. This is because many elements in parts of the design unreachable for a given change have some degree of textual similarity with the change statements. In the future, we intend to make the change statements more structured, e.g., by introducing a controlled natural language. This can make change impact analysis more targeted and deal with more complex situations.

# Chapter 8

## Conclusion

This chapter summarizes the research contributions of this dissertation and discusses potential areas for future work.

### 8.1 Summary

In this dissertation, we proposed several solutions for automating NL requirements analysis tasks using NLP. Our solutions build around requirements written as IEEE-830 style “shall” statements. Such requirements, in our experience, are very common in industry, including at our collaborating industry partners. We anticipate our solutions to be largely generalizable to other NL requirements formats, although a thorough investigation remains necessary. We have empirically evaluated all our solutions using real case studies and in collaboration with industry partners. Where possible, we have further employed interview surveys to assess the perceptions of practitioners about our proposed solutions.

In short, this dissertation made the following contributions:

Chapter 3 described an approach for checking conformance to requirements templates and a tool, named RETA, that we have built in support of the approach. We have evaluated our approach on four industrial case studies from different domains. As a part of our evaluation, we assessed and compared several text chunking solutions for their effectiveness in terms of accuracy for template conformance checking. Our results indicate that text chunking, along with syntactic parsing (when required), form an effective solution for conformance checking. The results further suggest that text chunking has little sensitivity to the presence or absence of a requirements glossary. This characteristic makes our solution applicable in early stages of requirements writing where a glossary may be unavailable.

Chapter 4 presented an approach for extracting candidate glossary terms and grouping these terms into clusters based on relatedness. The approach is supported by a tool named REGICE. One of the main advantages of our approach is that it provides guidelines on how to tune clustering for a given requirements document; this is important for a successful application of our approach in industry. We have applied our approach to three industrial case studies, in the context of which we have evaluated the accuracy and usefulness of the approach. An important finding from our evaluation is that, over requirements documents, our glossary term extraction technique is significantly more

accurate than generic term extraction tools. Furthermore, the feedback we collected from subject matter experts in our case studies suggests that our clustering technique offers practical benefits. Particularly, the experts found the resulting clusters helpful for better handling of some important tasks involved in the construction of requirements glossaries, including writing definitions for glossary terms and identifying the related terms.

Chapter 5 presented an automated approach for extracting domain models from requirements statements. The main technical contribution of our approach is in extending the existing set of model extraction rules, addressing major incompleteness issues in existing extractors. We have evaluated our approach over four industrial requirements documents, and have conducted a user interview survey with an expert from one of these case studies. Our evaluation contributes insights into the as yet limited knowledge available about the effectiveness of model extraction in industrial settings. An important observation from our evaluation is that a sizable fraction of automatically-extracted relations, despite being meaningful, are not relevant to the domain model. This observation can open avenues for future investigations on automated model extraction.

Chapter 6 presented our approach for analyzing the impact of changes in requirements documents, and our tool, named NARCIA, which we have developed to support this task. The key characteristic of our approach is that it exploits the phrasal structure of requirements sentences for analysis. Our evaluation indicates that our approach is accurate and practical over industrial requirements documents. An important hypothesis underlying our approach is that the large majority of requirements dependencies manifest themselves within the constituent phrases of requirements sentences. Across the change scenarios in our case studies, we could detect 99% (105 / 106) of the impacted requirements through phrasal analysis. Nevertheless, certain dependencies, as exemplified in our evaluation of Chapter 6, are inherently tacit and detectable only through explicit guidance.

Chapter 7 presented our approach for automatically identifying the impact of requirements changes on system designs expressed in Systems Modeling Language (SysML). For a given change, our approach computes a list of potentially impacted model elements by considering the reachability of elements using inter-block data flow, and intra-block control and data flow dependencies. Further, we rank the potentially impacted model elements based on the textual content of the model elements. Our evaluation on an industrial system shows that the accuracy of our approach consistently improves when we consider both inter-block and intra-block dependencies rather than only the inter-block ones, and the textual content of the diagrams in addition to only the elements' dependencies. Our approach further provides guidelines for traversing only a small fraction of the ranked list to detect the impacted model elements.

## 8.2 Future Work

In this dissertation, we focused on analyzing requirements written as “shall” statements. In the future, it is important to tailor our solutions to other NL requirements formats, such as use case descriptions. Furthermore, our solutions consider requirements statements one at a time when analyzing them. Other potentially-useful information in requirements documents, e.g., the sectioning and the sequence of the requirements statements, is not currently utilized. Considering such information might bring about further improvements, particularly for domain model extraction and change impact analysis.

Further user studies also remain necessary for better assessing the usefulness and applicability of our solutions.

An interesting technical improvement that we would like to consider across all our solutions in the future is the ability to dynamically adjust automatically-computed results in response to human feedback. With the current advancements in machine learning, it is foreseeable that a human feedback loop can be built into our solutions, with a machine learner learning from the experts' decisions in each step of an inspection and using the learned knowledge for improving the automation results. For example, while reviewing the clusters of glossary terms produced by our approach in Chapter 4, a machine learning algorithm can learn the characteristics of clusters that are rejected or significantly modified by the experts, in order to avoid producing clusters with similar characteristics.

Finally, we plan to explore the possibility of combining all the solutions in this dissertation into a commercial tool suite. Such a tool suite can, for example, be deployed as an add-in for commonly-used requirements management platforms such as IBM DOORS [DOORS, 2016]. Our industry partner, SES, has expressed interest in helping us pursue the commercialization of this dissertation's outcomes.



# List of Papers

Published papers included in this dissertation:

- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. "Automatic Checking of Conformance to Requirement Boilerplates via Text Chunking: An Industrial Case Study." at *7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 35-44. IEEE, 2013.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. "RUBRIC: A flexible tool for automated checking of conformance to requirement boilerplates." at *9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE – Tool Track)*, pp. 599-602. ACM, 2013.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Requirement Boilerplates: Transition From Manually-Enforced to Automatically-Verifiable Natural Language Patterns." at *4th IEEE International Workshop on Requirements Patterns (RePa)*, pp. 1-8. IEEE, 2014.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Improving requirements glossary construction via clustering: approach and industrial case studies." at *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 18:1-18:10. ACM, 2014.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Automated checking of conformance to requirements templates using natural language processing." In *IEEE transactions on Software Engineering (TSE)*, vol. 41-10, pp. 944-968. IEEE, 2015.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Change impact analysis for natural language requirements: An NLP approach." at *23rd IEEE International Requirements Engineering Conference (RE)*, pp. 6-15. IEEE, 2015.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "NARCIA: an automated tool for change impact analysis in natural language requirements." at *10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE – Tool Track)*, pp. 962-965. ACM, 2015.
- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Extracting Domain Models from Natural-Language Requirements: Approach and Industrial Evaluation." at *19th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, IEEE, 2016.
- Shiva Nejati, Mehrdad Sabetzadeh, Chetan Arora, Lionel Briand, and Felix Mandoux. "Automated Change Impact Analysis between SysML Models of Requirements and Design." at *24th*

*ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE),*  
ACM, 2016.

Unpublished papers included in this dissertation:

- Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. "Automated Extraction and Clustering of Requirements Glossary Terms." In *IEEE transactions on Software Engineering (TSE)*, (accepted pending minor revisions), 2016.

# Bibliography

[Abbott, 1983] Abbott, R. J. (1983). Program design by informal English descriptions. *Communications of the ACM*, 26(11).

[Achananuparp et al., 2008] Achananuparp, P., Hu, X., and Shen, X. (2008). The evaluation of sentence similarity measures. In Song, I.-Y., Eder, J., and Nguyen, T., editors, *Data Warehousing and Knowledge Discovery*, volume 5182 of *Lecture Notes in Computer Science*. Springer.

[Adedjouma et al., 2014] Adedjouma, M., Sabetzadeh, M., and Briand, L. (2014). Automated detection and resolution of legal cross references: Approach and a study of Luxembourg's legislation. In *22nd IEEE International Requirements Engineering Conference (RE'14)*, pages 63–72.

[Aggarwal and Reddy, 2013] Aggarwal, C. C. and Reddy, C. K. (2013). *Data clustering: algorithms and applications*. CRC Press.

[Aguilera and Berry, 1990] Aguilera, C. and Berry, D. (1990). The use of a repeated phrase finder in requirements extraction. *Journal of Systems and Software*, 13(3):209–230.

[Akbik and Broß, 2009] Akbik, A. and Broß, J. (2009). Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns. In *Workshop on Semantic Search at the 18th International World Wide Web Conference (WWW'09)*.

[Alomari et al., 2012] Alomari, H. W., Collard, M. L., and Maletic, J. I. (2012). A very efficient and scalable forward static slicing approach. In *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*, pages 425–434.

[Ambler, 2004] Ambler, S. (2004). *The Object Primer: Agile Model-Driven Development with UML 2.0*. Cambridge University Press.

[Ambriola and Gervasi, 2006] Ambriola, V. and Gervasi, V. (2006). On the systematic analysis of natural language requirements with CIRCE. *Automated Software Engineering*, 13(1).

[Amigó et al., 2009] Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. *IR*, 12(4).

[Amyot, 2003] Amyot, D. (2003). Introduction to the user requirements notation: learning by example. *Computer Networks*, 42(3).

[ANNIE, 2016] ANNIE (2016). GATE ANNIE: a Nearly-New Information Extraction System. <http://gate.ac.uk/sale/tao/splitch6.html>. Last accessed: August 2016.

## Bibliography

---

[Arafeen and Do, 2013] Arafeen, M. J. and Do, H. (2013). Test case prioritization using requirements-based clustering. In *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. IEEE.

[Arora et al., 2014a] Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2014a). Improving requirements glossary construction via clustering: Approach and industrial case studies. In *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2014)*, pages 18:1–18:10.

[Arora et al., 2014b] Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2014b). Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns. In *4th International Workshop on Requirements Patterns (RePa)*, pages 1–8.

[Arora et al., 2015a] Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2015a). Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41(10).

[Arora et al., 2016] Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2016). Extracting domain models from natural-language requirements: Approach and industrial evaluation. In *19th International Conference on Model Driven Engineering Languages and Systems (Models'16)*.

[Arora et al., 2013a] Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., and Gnaga, R. (2013a). Automatic checking of conformance to requirement boilerplates via text chunking: An industrial case study. In *ESEM'13*, pages 35–44.

[Arora et al., 2013b] Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., and Gnaga, R. (2013b). RUBRIC: A flexible tool for automated checking of conformance to requirement boilerplates. In *9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, pages 599–602.

[Arora et al., 2015b] Arora, C., Sabetzadeh, M., Goknil, A., Briand, L., and Zimmer, F. (2015b). Change impact analysis for natural language requirements: An NLP approach. In *23rd IEEE International Requirements Engineering Conference (RE'15)*.

[Arora et al., 2015c] Arora, C., Sabetzadeh, M., Goknil, A., Briand, L. C., and Zimmer, F. (2015c). NARCIA: an automated tool for change impact analysis in natural language requirements. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 962–965.

[Aryani et al., 2010] Aryani, A., Peake, I., and Hamilton, M. (2010). Domain-based change propagation analysis: An enterprise system case study. In *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*, pages 1–9.

[Attardi and Dell'Orletta, 2008] Attardi, G. and Dell'Orletta, F. (2008). Chunking and dependency parsing. In *Workshop on Partial Parsing: Between Chunking and Deep Parsing at 6th International Conference on Language Resources and Evaluation (LREC'08)*.

[Aubin and Hamon, 2006] Aubin, S. and Hamon, T. (2006). Improving term extraction with terminological resources. In *Advances in Natural Language Processing*. Springer.

[Banerjee and Pedersen, 2003] Banerjee, S. and Pedersen, T. (2003). Extended gloss overlaps as a measure of semantic relatedness. In *IJCAI*, volume 3.

[Barker and Cornacchia, 2000] Barker, K. and Cornacchia, N. (2000). Using noun phrase heads to extract document keyphrases. In *Advances in Artificial Intelligence*. Springer.

[Ben Abdessalem Karaa et al., 2015] Ben Abdessalem Karaa, W., Ben Azzouz, Z., Singh, A., Dey, N., S Ashour, A., and Ben Ghazala, H. (2015). Automatic builder of class diagram (ABCD): an application of UML generation from functional requirements. *Software: Practice and Experience*.

[Berry et al., 2003] Berry, D., Kamsties, E., and Krieger, M. (2003). From contract drafting to software specification: Linguistic sources of ambiguity, a handbook.

[Bird et al., 2009] Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly.

[Bock, 2006] Bock, C. (2006). SysML and UML 2 support for activity modeling. *Systems Engineering*, 9(2):160–186.

[Bohner and Arnold, 1996] Bohner, S. and Arnold, R. (1996). *Software Change Impact Analysis*. Wiley-IEEE, 1st edition.

[Bourigault, 1992] Bourigault, D. (1992). Surface grammatical analysis for the extraction of terminological noun phrases. In *14th Conf. on Computational Linguistics*.

[Bourigault and Jacquemin, 1999] Bourigault, D. and Jacquemin, C. (1999). Term extraction + term clustering: An integrated platform for computer-aided terminology. In *9th conference on European chapter of the Association for Computational Linguistics (EACL'99)*, pages 15–22.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1st edition.

[Briand et al., 2014] Briand, L. C., Falessi, D., Nejati, S., Sabetzadeh, M., and Yue, T. (2014). Traceability and SysML design slices to support safety inspections: A controlled experiment. *ACM Trans. Softw. Eng. Methodol.*, 23(1):9:1–9:43.

[Briand et al., 2003] Briand, L. C., Labiche, Y., and O'sullivan, L. (2003). Impact analysis and change management of UML models. In *International Conference on Software Maintenance (ICSM'03)*, pages 256–265.

[Buckland and Gey, 1994] Buckland, M. K. and Gey, F. C. (1994). The relationship between recall and precision. *Journal of the American Society for Information Science*, 45(1):12–19.

[Chantree et al., 2006] Chantree, F., Nuseibeh, B., De Roeck, A., and Willis, A. (2006). Identifying nocuous ambiguities in natural language requirements. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 59–68.

[Chemuturi, 2012] Chemuturi, M. (2012). *Requirements engineering and management for software development projects*. Springer Science & Business Media.

## Bibliography

---

[Chen et al., 2005] Chen, K., Zhang, W., Zhao, H., and Mei, H. (2005). An approach to constructing feature models based on requirements clustering. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*.

[Chen, 1983] Chen, P. P. (1983). English sentence structure and entity-relationship diagrams. *Information Sciences*, 29(2).

[Cleland-Huang, 2012] Cleland-Huang, J. (2012). Traceability in agile projects. In Cleland-Huang, J., Gotel, O., and Zisman, A., editors, *Software and Systems Traceability*. Springer.

[Cleland-Huang et al., 2007] Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R., and Romanova, E. (2007). Best practices for automated traceability. *Computer*, 40(6).

[Cleland-Huang et al., 2014] Cleland-Huang, J., Gotel, O., Huffman Hayes, J., Mäder, P., and Zisman, A. (2014). Software traceability: Trends and future directions. In *Future of Software Engineering (FOSE'14)*, pages 55–69.

[Cleland-Huang et al., 2005a] Cleland-Huang, J., Settimi, R., Benkhadra, O., Berezhanskaya, E., and Christina, S. (2005a). Goal-centric traceability for managing non-functional requirements. In *Proceedings of the 27th International Conference on Software Engineering, ICSE '05*, pages 362–371.

[Cleland-Huang et al., 2005b] Cleland-Huang, J., Settimi, R., Khadra, O. B., Berezhanskaya, E., and Christina, S. (2005b). Goal-centric traceability for managing non-functional requirements. In *ICSE*.

[Cohen, 1960] Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(01).

[Cohen et al., 2003] Cohen, W., Ravikumar, P., and Fienberg, S. (2003). A comparison of string distance metrics for name-matching tasks. In *Wrkshp. on Information Integration on the Web*, pages 73–78.

[Cormen et al., 2009] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, 3rd edition.

[Cradle, 2016] Cradle (2016). The Cradle Tool. <http://www.threesl.com/>. Last accessed: August 2016.

[Dahlstedt and Persson, 2005] Dahlstedt, Å. and Persson, A. (2005). Requirements interdependencies: State of the art and future challenges. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*. Springer.

[Daille, 2005] Daille, B. (2005). Variations and application-oriented terminology engineering. *Terminology*, 11(1).

[Daramola et al., 2012] Daramola, O., Sindre, G., and Stalhane, T. (2012). Pattern-based security requirements specification using ontologies and boilerplates. In *2nd International Workshop on Requirements Patterns (RePa'12)*, pages 54–59.

[de Gea et al., 2012] de Gea, J. M. C., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., and Vizcaíno, A. (2012). Requirements engineering tools: Capabilities, survey and assessment. *Information & Software Technology*, 54(10):1142–1157.

[De Marneffe and Manning, 2008] De Marneffe, M. C. and Manning, C. D. (2008). Stanford typed dependencies manual. Technical report, Stanford University.

[Deeptimahanti and Sanyal, 2011] Deeptimahanti, D. K. and Sanyal, R. (2011). Semi-automatic generation of UML models from natural language requirements. In *4th India Software Engineering Conference, ISEC '11*, pages 165–174. ACM.

[Delphi, 2016] Delphi (2016). Delphi automotive systems. <http://www.delphi.com/>.

[Dice, 1945] Dice, L. R. (1945). Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

[Dit et al., 2013] Dit, B., Revelle, M., Gethers, M., and Poshyvanyk, D. (2013). Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95.

[Divakaran, 2009] Divakaran, A. (2009). *Multimedia content analysis: theory and applications*. Springer Science & Business Media.

[DOORS, 2016] DOORS, I. (2016). [www.ibm.com/software/products/ca/en/ratidoor](http://www.ibm.com/software/products/ca/en/ratidoor).

[Drouin, 2003] Drouin, P. (2003). Term extraction using non-technical corpora as a point of leverage. *Terminology*, 9(1).

[Duan and Cleland-Huang, 2007] Duan, C. and Cleland-Huang, J. (2007). Clustering support for automated tracing. In *ASE'07*, pages 244–253.

[Dutoit et al., 2006] Dutoit, A., McCall, R., Mistrik, I., and Paech, B. (2006). *Rationale Management in Software Engineering*. Springer.

[Dwarakanath et al., 2013] Dwarakanath, A., Ramnani, R., and Sengupta, S. (2013). Automatic extraction of glossary terms from natural language requirements. In *RE*, pages 314–319.

[EA, 2016] EA (2016). Enterprise Architect (EA). <http://www.sparxsystems.com.au/>.

[Ebben, 2002] Ebben, P. (2002). Requirements for the WASP application platform. Technical Report WASP/D2.1, Telematica Instituut.

[Efron and Tibshirani, 1994] Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

[Elbendak et al., 2011] Elbendak, M., Vickers, P., and Rossiter, N. (2011). Parsed use case descriptions as a basis for object-oriented class model generation. *Journal of Systems and Software*, 84(7).

[Fabbrini et al., 2001] Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. (2001). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In *Proceedings of 26th Annual NASA Goddard Software Engineering Workshop (SEW'01)*, pages 97 – 105.

## Bibliography

---

[Fader et al., 2011] Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *Conference on Empirical Methods in Natural Language Processing*.

[Falessi et al., 2013] Falessi, D., Cantone, G., and Canfora, G. (2013). Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering*, 39(1).

[Farfeleder et al., 2011] Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Zojer, H., and Panis, C. (2011). DODT: Increasing requirements formalism using domain ontologies for improved embedded systems development. In *Proceedings of 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS'11)*, pages 271–274.

[Fellbaum, 1999] Fellbaum (1999). WordNet. <http://wordnet.princeton.edu>. Last accessed: August 2016.

[Femmer et al., 2014] Femmer, H., Méndez Fernández, D., Juergens, E., Klose, M., Zimmer, I., and Zimmer, J. (2014). Rapid requirements checks with requirements smells: Two case studies. In *1st International Workshop on Rapid Continuous Software Engineering (RCoSE'14)*, pages 10–19.

[Ferrari et al., 2013] Ferrari, A., Gnesi, S., and Tolomei, G. (2013). Using clustering to improve the structure of natural language requirements documents. In *Requirements Engineering: Foundation for Software Quality*. Springer.

[Fraley and Raftery, 2012] Fraley, C. and Raftery, A. (2012). mclust. <http://www.stat.washington.edu/mclust/>. Last accessed: August 2016.

[Francis and Kucera, 1982] Francis, W. and Kucera, H. (1982). *Frequency analysis of English usage*. Houghton Mifflin.

[Frantzi et al., 2000] Frantzi, K., Ananiadou, S., and Mima, H. (2000). Automatic recognition of multi-word terms: the C-value/NC-value method. *International Journal on Digital Libraries*, 3(02).

[Fraser, 2012] Fraser (2012). Diff, match and patch libraries for plain text. <http://code.google.com/p/google-diff-match-patch/>. <http://code.google.com/p/google-diff-match-patch/>.

[Friedenthal et al., 2008] Friedenthal, S., Moore, A., and Steiner, R. (2008). *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann.

[GATE, 2016] GATE (2016). GATE NLP Workbench. <http://gate.ac.uk/>. Last accessed: August 2016.

[GATE User Guide, 2016] GATE User Guide (2016). Developing Language Processing Components with GATE Version 8 (a User Guide). <http://gate.ac.uk/sale/tao/tao.pdf>. Last accessed: August 2016.

[Gervasi and Nuseibeh, 2002] Gervasi, V. and Nuseibeh, B. (2002). Lightweight validation of natural language requirements. *Software: Practice and Experience*, 32(2):113–133.

[Gervasi and Zowghi, 2005] Gervasi, V. and Zowghi, D. (2005). Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology*, 14(3):277–330.

[Goknil et al., 2014a] Goknil, A., Kurtev, I., van den Berg, K., and Spijkerman, W. (2014a). Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, 56(8):950–972.

[Goknil et al., 2014b] Goknil, A., van Domburg, R., Kurtev, I., van den Berg, K., and Wijnhoven, F. (2014b). Experimental evaluation of a tool for change impact prediction in requirements models. In *MoDRE’14*.

[Goldin and Berry, 1997] Goldin, L. and Berry, D. (1997). AbstFinder: a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, 4(04).

[Gomaa and Fahmy, 2013] Gomaa, W. and Fahmy, A. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13).

[Gregory, 2011] Gregory, S. (2011). Easy EARS: Rapid application of the easy approach to requirements syntax. In *19th IEEE International Requirements Engineering Conference (RE’11)*. Tutorial.

[Güldali et al., 2009] Güldali, B., Sauer, S., Engels, G., Funke, H., and Jahnich, M. (2009). Semi-automated test planning for e-ID systems by using requirements clustering. In *24th IEEE/ACM International Conference on Automated Software Engineering (ASE’09)*, pages 29–39.

[Guzman and Maalej, 2014] Guzman, E. and Maalej, W. (2014). How do users like this feature? a fine grained sentiment analysis of app reviews. In *22nd IEEE International Requirements Engineering Conference (RE’14)*, pages 153–162.

[Harmain and Gaizauskas, 2003] Harmain, H. and Gaizauskas, R. (2003). CM-Builder: A natural language-based CASE tool for object-oriented analysis. *Automated Software Engineering*, 10(2).

[Hepple, 2000] Hepple, M. (2000). Independence and commitment: assumptions for rapid training and execution of rule-based POS taggers. In *38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, pages 278–277.

[Heylen and De Hertog, 2015] Heylen, K. and De Hertog, D. (2015). Automatic term extraction. *Handbook of Terminology*, 1.

[Hirst and St-Onge, 1998] Hirst, G. and St-Onge, D. (1998). Lexical chains as representations of context for the detection and correction of malapropisms. In Fellbaum, C., editor, *WordNet: An electronic lexical database*, pages 305–332. MIT Press.

[Holbrook et al., 2009] Holbrook, E., Hayes, J., and Dekhtyar, A. (2009). Toward automating requirements satisfaction assessment. In *17th IEEE International Requirements Engineering Conference (RE’09)*, pages 149–158.

[Holt and Perry, 2008] Holt, J. and Perry, S. (2008). *SysML for Systems Engineering: Institute of Engineering and Technology*. IET Digital Library.

## Bibliography

---

[Holt et al., 2011] Holt, J., Perry, S., and Brownsword, M. (2011). *Model-Based Requirements Engineering*. IET.

[Hull et al., 2011] Hull, M., Jackson, K., and Dick, J. (2011). *Requirements Engineering*. Springer, 3rd edition.

[Ibrahim and Ahmad, 2010] Ibrahim, M. and Ahmad, R. (2010). Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *2nd International Conference on Computer Research and Development (ICCRD'10)*.

[IEC, 2005] IEC (2005). Functional safety of electrical / electronic / programmable electronic safety-related systems (IEC 61508). International Electrotechnical Commission: International Electrotechnical Commission.

[IEEE Computer Society and Board, 1998] IEEE Computer Society, S. E. S. C. and Board, I.-S. S. (1998). Ieee recommended practice for software requirements specifications. *IEEE Std 830-1998*.

[INCOSE, 2016] INCOSE (2016). International Council on Systems Engineering. <http://www.incose.org/>.

[Indurkhy and Damerau, 2010] Indurkhy, N. and Damerau, F. J. (2010). *Handbook of natural language processing*. CRC Press.

[inteGREAT, 2016] inteGREAT (2016). The inteGREAT Tool. <http://www.edevtech.com>. last accessed: August 2016.

[ISO26262, 2009] ISO26262 (2009). Road vehicles – functional safety. ISO draft standard.

[Jiang and Conrath, 1997] Jiang, J. J. and Conrath, D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc of 10th International Conference on Research in Computational Linguistics, ROCLING'97*.

[Jones et al., 1990] Jones, L., Gassie, E., and Radhakrishnan, S. (1990). INDEX: The statistical basis for an automatic conceptual phrase-indexing system. *Journal of the American Society for Information Science and Technology*, 41(02).

[Jönsson and Lindvall, 2005] Jönsson, P. and Lindvall, M. (2005). Impact analysis. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*. Springer.

[Joseph et al., 2013] Joseph, S., Schumm, M., Rummel, O., Soska, A., Reschke, M., Mottok, J., Niemetz, M., and Schroll-Decker, I. (2013). Teaching finite state machines with case method and role play. In *IEEE Global Engineering Education Conference (EDUCON'13)*, pages 1305–1312.

[Jurafsky and Martin, 2009] Jurafsky, D. and Martin, J. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2nd edition.

[Justeson and Katz, 1995] Justeson, J. S. and Katz, S. M. (1995). Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1(01).

[Kiyavitskaya et al., 2008a] Kiyavitskaya, N., Zeni, N., Breaux, T., Antón, A., Cordy, J., Mich, L., and Mylopoulos, J. (2008a). Automating the extraction of rights and obligations for regulatory compliance. In *27th International Conference on Conceptual Modeling (ER'08)*, pages 154–168.

[Kiyavitskaya et al., 2008b] Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. (2008b). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, 13(3).

[Klein and Manning, 2016] Klein, D. and Manning, C. D. (2016). The Stanford Parser: A statistical parser. <http://nlp.stanford.edu/software/lex-parser.shtml>. Last accessed: August 2016.

[Kof et al., 2010] Kof, L., Gacitua, R., Rouncefield, M., and Sawyer, P. (2010). Ontology and model alignment as a means for requirements validation. In *4th IEEE International Conference on Semantic Computing (ICSC'10)*, pages 46–51.

[Kuang et al., 2012] Kuang, H., Mäder, P., Hu, H., Ghabi, A., Huang, L., Lv, J., and Egyed, A. (2012). Do data dependencies in source code complement call dependencies for understanding requirements traceability? In *Proceedings of 28th IEEE International Conference on Software Maintenance (ICSM'12)*, pages 181–190.

[Larman, 2005] Larman, C. (2005). *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. Prentice Hall.

[Lauesen, 2002] Lauesen, S. (2002). *Software Requirements: Styles & Techniques*. Addison-Wesley, 1st edition.

[Leacock and Chodorow, 1998] Leacock, C. and Chodorow, M. (1998). Combining local context and wordnet similarity for word sense identification. In Fellbaum, C., editor, *WordNet: An electronic lexical database*, pages 265–283. MIT Press.

[Lehnert, 2011] Lehnert, S. (2011). A review of software change impact analysis. In *Technical Report - Technische Universität Ilmenau*.

[LEXIOR, 2016] LEXIOR (2016). The LEXIOR Tool. <http://www.cortim.com/lexior/staticdemo/>. last accessed: August 2016.

[Li et al., 2013] Li, B., Sun, X., Leung, H., and Zhang, S. (2013). A survey of code-based change impact analysis techniques. *Softw. Test., Verif. Reliab.*, 23(8):613–646.

[Likert, 1932] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*, 22(140).

[Lin, 1998] Lin, D. (1998). An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304.

[Liu et al., 2004] Liu, D., Subramaniam, K., Eberlein, A., and Far, B. H. (2004). Natural language requirements analysis and class model generation using UCDA. In *Innovations in Applied Artificial Intelligence*. Springer.

## Bibliography

---

[Liu et al., 2003] Liu, D., Subramaniam, K., Far, B. H., and Eberlein, A. (2003). Automating transition from use-cases to class model. In *Canadian Conference on Electrical and Computer Engineering (CCECE'03)*.

[Maalej and Nabil, 2015] Maalej, W. and Nabil, H. (2015). Bug report, feature request, or simply praise? on automatically classifying app reviews. In *23rd IEEE International Requirements Engineering Conference (RE'15)*, pages 116–125.

[Mäder et al., 2013] Mäder, P., Jones, P., Zhang, Y., and Cleland-Huang, J. (2013). Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66.

[Mahmoud, 2015] Mahmoud, A. (2015). An information theoretic approach for extracting and tracking non-functional requirements. In *RE'15*.

[Manning et al., 2008] Manning, C., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge, 1st edition.

[Manning and Schütze, 1999] Manning, C. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[Marcus et al., 1993] Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[Martorell et al., 2014] Martorell, S., Soares, C. G., and Barnett, J. (2014). *Safety, Reliability and Risk Analysis: Theory, Methods and Applications*. CRC Press.

[Matsuo and Ishizuka, 2004] Matsuo, Y. and Ishizuka, M. (2004). Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13.

[Mavin, 2012] Mavin, A. (2012). Listen, then use EARS. *IEEE Software*, 29(2):17–18.

[Mavin and Wilkinson, 2010] Mavin, A. and Wilkinson, P. (2010). Big EARS (the return of “Easy Approach to Requirements Engineering”). In *18th IEEE International Requirements Engineering Conference (RE'10)*, pages 277–282.

[Mavin et al., 2009] Mavin, A., Wilkinson, P., Harwood, A., and Novak, M. (2009). Easy approach to requirements syntax (EARS). In *17th IEEE International Requirements Engineering Conference (RE'09)*, pages 317–322.

[MBSE, 2008] MBSE (2008). Survey of model-based systems engineering (MBSE) methodologies. INCOSE Survey.

[McGill and Salton, 1983] McGill, M. and Salton, G. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill.

[Ménard and Ratté, 2015] Ménard, P. A. and Ratté, S. (2015). Concept extraction from business documents for software engineering projects. *Automated Software Engineering*.

[Mich, 1996] Mich, L. (1996). NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. *Natural language engineering*, 2(02).

[Misra, 2015] Misra, J. (2015). Terminological inconsistency analysis of natural language requirements. *Information and Software Technology (IST)*. To appear.

[Monge and Elkan, 1997] Monge, A. and Elkan, C. (1997). Efficient domain-independent detection of approximately duplicate database records. In *Wrkshp. on Research Issues in on Knowledge Discovery and Data Mining*, pages 23–29.

[Müller and Rumpe, 2014] Müller, K. and Rumpe, B. (2014). A model-based approach to impact analysis using model differencing. *Electronic Communications of the EASST*, 65.

[MuNPEx, 2016] MuNPEx (2016). Multilingual Noun Phrase Extractor (MuNPEx). Last accessed: August 2016.

[Nejati et al., 2016] Nejati, S., Sabetzadeh, M., Arora, C., Briand, L., and Mandoux, F. (2016). Automated change impact analysis between sysml models of requirements and design. In *24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Seattle 13-18 November 2016*.

[Nejati et al., 2012] Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., and Zave, P. (2012). Matching and merging of variant feature specifications. *IEEE Transactions on Software Engineering*, 38(06).

[OMG, 2016] OMG (2016). Systems modeling language. <http://www.omg.sysml.org/>.

[OpenNLP, 2016] OpenNLP (2016). Apache’s OpenNLP. <http://opennlp.apache.org>. last accessed: August 2016.

[Pazienza et al., 2005] Pazienza, M., Pennacchiotti, M., and Zanzotto, F. (2005). Terminology extraction: an analysis of linguistic and statistical approaches. In *Knowledge Mining*. Springer.

[Pedersen et al., 2004] Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). Wordnet::similarity - measuring the relatedness of concepts. In *19th National Conference on Artificial Intelligence (AAAI’04)*, pages 1024–1025.

[Pfleeger and Atlee, 2009] Pfleeger, S. L. and Atlee, J. M. (2009). *Software engineering - theory and practice (4. ed.)*. Pearson Education.

[Pirkola et al., 1999] Pirkola, A., Keskustalo, H., and Järvelin, K. (1999). The effects of conjunction, facet structure, and dictionary combinations in concept-based cross-language retrieval. *IR*, 1(3).

[Pohl, 1996] Pohl, K. (1996). *Process-Centered Requirements Engineering*. Wiley.

[Pohl, 2010] Pohl, K. (2010). *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, 1st edition.

[Pohl and Rupp, 2011] Pohl, K. and Rupp, C. (2011). *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level –IREB compliant*. Rocky Nook, 1st edition.

## Bibliography

---

[Popescu et al., 2008] Popescu, D., Rugaber, S., Medvidovic, N., and Berry, D. (2008). Reducing ambiguities in requirements specifications via automatically created object-oriented models. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, volume 5320. Springer.

[Pruski et al., 2015] Pruski, P., Lohar, S., Goss, W., Rasin, A., and Cleland-Huang, J. (2015). Tiqi: answering unstructured natural language trace queries. *Requirements Engineering*, 20(03).

[R, 2016] R (2016). The R project. <http://www.r-project.org/>. Last accessed: August 2016.

[Ramshaw and Marcus, 1995] Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In *Proceedings of the 3rd ACL Workshop on Very Large Corpora*, pages 82–94.

[Rempel et al., 2014] Rempel, P., Mäder, P., Kuschke, T., and Cleland-Huang, J. (2014). Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 943–954.

[Resnik, 1995] Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95. Morgan Kaufmann Publishers Inc.

[Riaz et al., 2014] Riaz, M., King, J., Slankas, J., and Williams, L. (2014). Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *22nd IEEE International Requirements Engineering Conference (RE'14)*, pages 183–192.

[Rolland and Proix, 1992] Rolland, C. and Proix, C. (1992). A natural language approach for requirements engineering. In *Proceedings of 4th Conference on Advanced Information Systems Engineering (CAiSE'92)*, pages 257–277.

[Rosenberg and Hirschberg, 2007] Rosenberg, A. and Hirschberg, J. (2007). V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL*, volume 7.

[RQA, 2016] RQA (2016). RQA: The Requirements Quality Analyzer Tool. <http://www.reusecompany.com/rqa>. last accessed: August 2016.

[Rupp, 2009] Rupp, C. (2009). *Requirements-Engineering und -Management: professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser.

[Rus et al., 2013] Rus, V., Lintean, M., Banjade, R., Niraula, N., and Stefanescu, D. (2013). SEMI-LAR: The semantic similarity toolkit. In *ACL*, pages 163–168.

[Sabetzadeh et al., 2011] Sabetzadeh, M., Nejati, S., Briand, L. C., and Mills, A. E. (2011). Using SysML for modeling of safety-critical software-hardware interfaces: Guidelines and industry experience. In *13th IEEE International Symposium on High-Assurance Systems Engineering, HASE 2011, Boca Raton, FL, USA, November 10-12, 2011*, pages 193–201.

[Schneider, 2009] Schneider, K. (2009). *Experience and Knowledge Management in Software Engineering*, chapter Structuring Knowledge for Reuse. Springer.

[Scholz, 1985] Scholz, F. (1985). Maximum likelihood estimation. In Kotz, S., Johnson, N., and Read, C., editors, *Encyclopedia of Statistical Sciences*. Wiley.

[Schwarz et al., 1978] Schwarz, G. et al. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6.

[SES, 2016] SES (2016). SES - Global Satellite Services Provider - Your Satellite Company. <http://www.ses.com>. Accessed Mar. 11, 2016.

[SimPack, 2016] SimPack (2016). SimPack: A generic Java library for similarity measures in ontologies. <http://www.ifi.uzh.ch/ddis/simpack.html>. Last accessed: August 2016.

[Slipper et al., 2013] Slipper, D., McEwan, A., and Ifill, W. (2013). A framework for specification of arming system safety functions. In *8th IET International System Safety and Cyber Security Conference*, pages 1–7.

[Smith, 2011] Smith, N. A. (2011). *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool.

[Sommerville and Sawyer, 1997] Sommerville, I. and Sawyer, P. (1997). *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc.

[Song et al., 2006] Song, M., Song, I., and Lee, K. (2006). Automatic extraction for creating a lexical repository of abbreviations in the biomedical literature. In Tjoa, A. and Trujillo, J., editors, *Data Warehousing and Knowledge Discovery*, volume 4081, pages 384–393. Springer.

[StanPOS, 2016] StanPOS (2016). Stanford Log-linear Part-Of-Speech Tagger. <http://nlp.stanford.edu/software/tagger.shtml>. Last accessed: August 2016.

[Stevens, 2006] Stevens, P. (2006). *Using UML: software engineering with objects and components*. Pearson Education.

[STUK, 2016] STUK (2016). List of regulatory guides on nuclear safety (YVL). Last accessed: September 2014.

[Sultanov and Hayes, 2010] Sultanov, H. and Hayes, J. (2010). Application of swarm techniques to requirements engineering: Requirements tracing. In *18th IEEE International Requirements Engineering Conference (RE'10)*, pages 211–220.

[Sundaram et al., 2010] Sundaram, S., Hayes, J., Dekhtyar, A., and Holbrook, E. (2010). Assessing traceability of software engineering artifacts. *Requirements Engineering*, 15(3).

[Tang et al., 2007] Tang, A., Nicholson, A., Jin, Y., and Han, J. (2007). Using bayesian belief networks for change impact analysis in architecture design. *J. Syst. Softw.*, 80(1):127–148.

[TermRaider, 2016] TermRaider (2016). TermRaider. <http://gate.ac.uk/projects/neon/>. Last accessed: August 2016.

[Terzakis, 2013] Terzakis, J. (2013). Reducing requirements defect density by using mentoring to supplement training. *International Journal On Advances in Intelligent Systems*, 6(1&2):102–111.

## Bibliography

---

[TextRank, 2016] TextRank (2016). TextRank. <http://github.com/ceteri/textrank/>. Last accessed: August 2016.

[TigerPro, 2016] TigerPro (2016). Tiger Pro: Tool to ingest and elucidate requirements. <http://www.therightrequirement.com/TigerPro/TigerPro.html>. last accessed: August 2016.

[Tip, 1995] Tip, F. (1995). A survey of program slicing techniques. *J. Prog. Lang.*, 3(3).

[TOPIA, 2016] TOPIA (2016). TOPIA. <http://pypi.python.org/pypi/topia.termextract/>. Last accessed: August 2016.

[Torkar et al., 2012] Torkar, R., Gorschek, T., Feldt, R., Svahnberg, M., Raja, U., and Kamran, K. (2012). Requirements traceability: a systematic review and industry case study. *International Journal of Software Engineering and Knowledge Engineering*, 22(3).

[Uusitalo et al., 2011] Uusitalo, E., Raatikainen, M., Mannisto, T., and Tommila, T. (2011). Structured natural language requirements in nuclear energy domain towards improving regulatory guidelines. In *4th International Workshop on Requirements Engineering and Law (RELAW'11)*, pages 67–73.

[van den Berg, 2006] van den Berg, K. (2006). Change impact analysis of crosscutting in software architectural design. In *Workshop on Architecture-Centric Evolution (ACE 2006)*, pages 1–15, Groningen.

[van Lamsweerde, 2009] van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 1st edition.

[Vidya Sagar and Abirami, 2014] Vidya Sagar, V. and Abirami, S. (2014). Conceptual modeling of natural language functional requirements. *Journal of Systems and Software*, 88(01):25–41.

[Visible-Thread, 2016] Visible-Thread (2016). The Visible Thread Tool. <http://www.visiblethread.com/>. last accessed: August 2016.

[Wiegers, 2005] Wiegers, K. (2005). *More about software requirements: thorny issues and practical advice*. Microsoft Press.

[Wilson et al., 1997] Wilson, W., Rosenberg, L., and Hyatt, L. (1997). Automated analysis of requirement specifications. In *Proceedings of 19th International Conference on Software Engineering (ICSE'97)*, pages 161–171.

[Withall, 2007] Withall, S. (2007). *Software Requirement Patterns (Best Practices)*. Microsoft Press, 1st edition.

[Wong and Cai, 2009] Wong, S. and Cai, Y. (2009). Predicting change impact from logical models. In *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*, pages 467–470.

[Wu and Palmer, 1994] Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. In *32nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics.

[Yang et al., 2011] Yang, H., De Roeck, A., Gervasi, V., Willis, A., and Nuseibeh, B. (2011). Analysing anaphoric ambiguity in natural language requirements. *Requir. Eng.*, 16(3):163–189.

[Yang et al., 2010] Yang, Z., Lin, H., and Li, Y. (2010). BioPPISVMExtractor: A protein–protein interaction extractor for biomedical literature using SVM and rich feature sets. *Journal of biomedical informatics*, 43(1).

[Yilmaz and Yilmaz, 2011] Yilmaz, A. E. and Yilmaz, I. B. (2011). Natural language processing techniques in requirements engineering. In Ramachandran, M., editor, *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*. IGI Global.

[Young, 2004] Young, R. (2004). *The Requirements Engineering Handbook*. Artech House.

[Yue et al., 2011] Yue, T., Briand, L., and Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models. *Requir. Eng.*, 16(2):75–99.

[Yue et al., 2015] Yue, T., Briand, L. C., and Labiche, Y. (2015). aToucan: An automated framework to derive UML analysis models from use case models. *ACM Transactions on Software Engineering and Methodology*, 24(3).

[Zachos and Maiden, 2008] Zachos, K. and Maiden, N. (2008). Inventing requirements from software: An empirical investigation with web services. In *16th IEEE International Requirements Engineering Conference (RE'08)*.

[Zeni et al., 2015] Zeni, N., Kiyavitskaya, N., Mich, L., Cordy, J., and Mylopoulos, J. (2015). GaiusT: Supporting the extraction of rights and obligations for regulatory compliance. *Requirements Engineering*, 20(01).

[Zhang et al., 2014] Zhang, H., Li, J., Zhu, L., Jeffery, D., Liu, Y., Wang, Q., and Li, M. (2014). Investigating dependencies in software requirements for change propagation analysis. *IST*, 56(1).

[Zhang et al., 2008] Zhang, Z., Iria, J., Brewster, C., and Ciravegna, F. (2008). A comparative evaluation of term recognition algorithms. In *6th Intl. Conf. on Lang. Resources and Evaluation*.

[Zhao et al., 2002] Zhao, J., Yang, H., Xiang, L., and Xu, B. (2002). Change impact analysis to support architectural evolution. *J. of Software Maintenance and Evolution*, 14(5).

[Zhu et al., 2013] Zhu, M., Zhang, Y., Chen, W., Zhang, M., and Zhu, J. (2013). Fast and accurate shift-reduce constituent parsing. In *51st Annual Meeting of the Association for Computational Linguistics (ACL'13)*.

[Zou et al., 2010] Zou, X., Settimi, R., and Cleland-Huang, J. (2010). Improving automated requirements trace retrieval: a study of term-based enhancement methods. *Empirical Software Engineering*, 15(2):119–146.