Why and How to Extract Conditional Statements From Natural Language Requirements

Inaugural-Dissertation zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von Jannik Fischbach aus Marburg

Köln, 2022

Berichterstatter (Gutachter):

1. Prof. Dr. Andreas Vogelsang, Universität zu Köln
2. Prof. Dr. Michael Felderer, Universität Innsbruck
3. Prof. Dr. Walid Maalej, Universität Hamburg

Tag der mündlichen Prüfung: 14.09.2022

Abstract

Functional requirements often describe system behavior by relating events to each other, e.g. *"If the system detects an error (e_1), an error message shall be shown (e_2)"*. Such conditionals consist of two parts: the *antecedent* (see e_1) and the *consequent* (e_2), which convey strong, semantic information about the intended behavior of a system. Automatically extracting conditionals from texts enables several analytical disciplines and is already used for information retrieval and question answering. We found that automated conditional extraction can also provide added value to *Requirements Engineering* (RE) by facilitating the automatic derivation of acceptance tests from requirements. However, the potential of extracting conditionals has not yet been leveraged for RE. We are convinced that this has two principal reasons:

① The extent, form, and complexity of conditional statements in RE artifacts is not well understood. We do not know how conditionals are formulated and logically interpreted by RE practitioners. This hinders the development of suitable approaches for extracting conditionals from RE artifacts.

② Existing methods fail to extract conditionals from *Unrestricted Natural Language* (NL) in fine-grained form. That is, they do not consider the combinatorics between *antecedents* and *consequents*. They also do not allow to split them into more fine-granular text fragments (e.g., variable and condition), rendering the extracted conditionals unsuitable for RE downstream tasks such as test case derivation.

This thesis contributes to both areas. In **Part I**, we present empirical results on the prevalence and logical interpretation of conditionals in RE artifacts. We found that conditionals in requirements mainly occur in explicit, marked form and may include up to three *antecedents* and two *consequents*. Hence, the extraction approach needs to understand conjunctions, disjunctions, and negations to fully capture the relation between *antecedents* and *consequents*. We also found that conditionals are a source of ambiguity and there is not just one way to interpret them formally. This affects any automated analysis that builds upon formalized requirements (e.g., inconsistency checking) and may also influence guidelines for writing requirements.

Part II presents our tool-supported approach `CiRA` capable of detecting conditionals in NL requirements and extracting them in fine-grained form. For the detection, `CiRA` uses syntactically enriched BERT embeddings combined with a softmax classifier and outperforms existing methods (macro-F_1: 82 %). Our experiments show that a sigmoid classifier built on RoBERTa embeddings is best suited to extract conditionals in fine-grained form (macro-F_1: 86 %). `CiRA` is available at `http://www.cira.bth.se/demo/`.

In **Part III**, we highlight how the extraction of conditionals from requirements can help to create acceptance tests automatically. First, we motivate this use case in an empirical study and demonstrate that the lack of adequate acceptance tests is one of the major problems in agile testing. Second, we show how extracted conditionals can be mapped to a *Cause-Effect-Graph* from which test cases can be derived automatically. We demonstrate the feasibility of our approach in a case study with three industry partners. In our study, out of 578 manually created test cases, 71.8 % can be generated automatically. Furthermore, our approach discovered 80 relevant test cases that were missed in manual test case design. **At the end of this thesis,** the reader will have an understanding of (1) the notion of conditionals in RE artifacts, (2) how to extract them in fine-grained form, and (3) the added value that the extraction of conditionals can provide to RE.

Acknowledgements

This thesis would not have been possible without the support of many people. First of all, I would like to thank my Ph.D. supervisor Prof. Dr. Andreas Vogelsang, who was always very supportive in any imaginable way. I am grateful for the flexibility Andreas gave me to look for a topic that matches my interests and to pursue it largely on my own. I owe Andreas a debt of gratitude for his guidance I got whenever I needed it. I also want to thank my co-supervisor Prof. Dr. Michael Felderer for valuable feedback on this thesis.

Furthermore, I would like to thank the whole Qualicen team - I learned a lot with and from you. Particularly, I would like to thank Dr. Maximilian Junker, Dr. Sebastian Eder, Dr. Henning Femmer, and Dr. Benedikt Hauptmann for their confidence in me. I will always be grateful to you for joining me on the Ph.D. journey in May 2019 and helping me to juggle research and work time. Many thanks also to Silke Müller and Andreas Horn for their constantly positive attitude, which helped me to keep the ball rolling even in challenging times.

Many of my papers were written in collaboration with colleagues, fellow researchers, and working students. I would like to thank Tobias Springer, Andreas Wehrle, Pablo Restrepo Henao, Lukas Konwitschny, Dominik Spies, Arjen Spaans, and Maximilian Kummeth for their commitment and support. I am especially grateful for the opportunity to work with Julian Frattini. Thank you Julian for your reliability and your 100 % engagement in the realization of ideas. I would also like to thank Yannick Debes, who reviewed my papers and encouraged me with his advice. Thank you so much for always being there for me when I need you.

I am also grateful for the privilege of spending a research period at the *Software Engineering Research and Education Lab* (SERL) Sweden. My special thanks go to Prof. Dr. Daniel Mendez who warmly welcomed me with open arms and always creates a pleasant atmosphere with his friendly, *"you can always talk to me"* attitude. Thanks also to Dr. Davide Fucci, Dr. Michael Unterkalmsteiner, and Prof. Dr. Krzysztof Wnuk for valuable insights into the life of a professor and advice for an academic career. I look forward to future research projects with you.

The close cooperation with industry partners helped me to understand many problems addressed in this thesis and to evaluate my approach(es) on real-world data. I would like to thank especially Jeannette Radduenz, Dietmar Freudenstein, Simone Gornig, Holger Kasperczyk (*Allianz Deutschland AG*), Parisa Yousefi, Tedi Juricic (*Ericsson*), and Carsten Wiecher (*Leopold Kostal GmbH & Co. KG*) for participating in my case studies.

Most of all, I would like to thank my family and friends for their unconditional support. I would not be standing where I am now without you. I am deeply grateful for my parents Inge and Michael, my brother Moritz and, above all, my girlfriend Katharina. I love you.

Publication Preface

Parts of the contributions presented in this thesis are based on previous publications:

[1]: J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, and D. Freudenstein, "Specmate: Automated creation of test cases from acceptance criteria," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 321–331, 2020, **acceptance rate: n.a.**

[2]: J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, "Towards causality extraction from requirements," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 388–393, 2020, **acceptance rate: 29 %**

[3]: J. Fischbach, H. Femmer, D. Mendez, D. Fucci, and A. Vogelsang, "What makes agile test artifacts useful? an activity-based quality model from a practitioners' perspective," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, (New York, NY, USA), Association for Computing Machinery, 2020, **acceptance rate: 12 %, awarded as best industry paper** 🏆

[4]: J. Fischbach, J. Frattini, A. Spaans, M. Kummeth, A. Vogelsang, D. Mendez, and M. Unterkalmsteiner, "Automatic detection of causality in requirement artifacts: The cira approach," in *Requirements Engineering: Foundation for Software Quality* (F. Dalpiaz and P. Spoletini, eds.), (Cham), pp. 19–36, Springer International Publishing, 2021, **acceptance rate: 27 %, awarded as best research paper** 🏆

[5]: J. Fischbach, J. Frattini, D. Mendez, M. Unterkalmsteiner, H. Femmer, and A. Vogelsang, "How do practitioners interpret conditionals in requirements?," in *Product-Focused Software Process Improvement* (L. Ardito, A. Jedlitschka, M. Morisio, and M. Torchiano, eds.), (Cham), pp. 85–102, Springer International Publishing, 2021, **acceptance rate: 39 %**

[6]: J. Fischbach, T. Springer, J. Frattini, H. Femmer, A. Vogelsang, and D. Mendez, "Fine-grained causality extraction from natural language requirements using recursive neural tensor networks," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 60–69, 2021, **acceptance rate: 67 %**

[7]: J. Frattini, J. Fischbach, D. Mendez, M. Unterkalmsteiner, A. Vogelsang, and K. Wnuk, "Causality in requirements artifacts: prevalence, detection, and impact," *Requirements Engineering*, Feb 2022, **extended version of [4]**

[8]: C. Wiecher, J. Fischbach, J. Greenyer, A. Vogelsang, C. Wolff, and R. Dumitrescu, "Integrated and iterative requirements analysis and test specification: A case study at kostal," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 112–122, 2021, **acceptance rate: 41 %**

[9]: N. Jadallah, J. Fischbach, J. Frattini, and A. Vogelsang, "Cate: Causality tree extractor from natural language requirements," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 77–79, 2021, **acceptance rate: 67 %**

Under Review

In addition, one paper included in this thesis is still under review. The author's drafts are included.

[10]: J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, and C. Wiecher, "Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study," 2022, Submitted to the In-Practice Track at the *Journal of Systems and Software*

Contribution Statement

Jannik Fischbach is the main author of most of the publications included in this thesis. As the lead author, he took the main responsibility for design, implementation, data collection and analysis, and publishing the results in peer-reviewed venues. His and the co-authors' contribution to the included publications is described next using the Contributor Roles Taxonomy (CRediT):

[1]: **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Vogelsang:** Methodology, Conceptualization, Writing - Review & Editing, Supervision. **Spies:** Conceptualization, Software, Writing - Review & Editing. **Wehrle:** Software. **Junker:** Supervision, Writing - Review & Editing. **Freudenstein:** Validation, Resources.

[2]: **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Hauptmann:** Conceptualization, Methodology, Writing - Review & Editing. **Konwitschny:** Software, Data Curation. **Spies:** Conceptualization. **Vogelsang:** Supervision, Writing - Review & Editing.

[3]: **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Femmer:** Conceptualization, Writing - Original Draft. **Mendez:** Conceptualization, Writing - Review & Editing, Supervision. **Fucci:** Methodology, Writing - Review & Editing. **Vogelsang:** Conceptualization, Writing - Review & Editing, Supervision.

[4]: **Fischbach:** Conceptualization, Methodology, Investigation, Software, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Frattini:** Conceptualization, Investigation, Software, Data Curation, Writing - Review & Editing. **Spaans:** Data Curation. **Kummeth:** Data Curation. **Vogelsang:** Conceptualization, Supervision. **Mendez:** Methodology, Writing - Review & Editing. **Unterkalmsteiner:** Methodology, Writing - Review & Editing.

[5]: **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Frattini:** Conceptualization, Investigation, Data Curation, Writing - Review & Editing, Visualization.

Mendez: Methodology, Writing - Review & Editing. **Unterkalmsteiner:** Methodology, Writing - Review & Editing. **Femmer:** Resources, Conceptualization, Writing - Review & Editing. **Vogelsang:** Supervision, Conceptualization, Writing - Review & Editing.

[6]: **Fischbach:** Conceptualization, Methodology, Investigation, Software, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Springer:** Software, Data Curation, Visualization. **Frattini:** Writing - Review & Editing, Conceptualization. **Femmer:** Conceptualization. **Vogelsang:** Conceptualization, Supervision. **Mendez:** Writing - Review & Editing, Conceptualization.

[7]: **Frattini:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Review & Editing, Visualization. **Mendez:** Supervision, Conceptualization, Writing - Review & Editing. **Unterkalmsteiner:** Methodology, Writing - Review & Editing. **Vogelsang:** Methodology, Supervision. **Wnuk:** Resources, Supervision.

[8]: **Wiecher:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Fischbach:** Conceptualization, Investigation, Data Curation, Writing - Review & Editing. **Greenyer:** Methodology, Writing - Review & Editing. **Vogelsang:** Supervision, Conceptualization. **Wolff:** Supervision, Conceptualization. **Dumitrescu:** Supervision, Conceptualization.

[9]: **Jadallah:** Software, Conceptualization, Data Curation, Writing - Original Draft, Writing - Review & Editing. **Fischbach:** Conceptualization, Methodology, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization. **Frattini:** Conceptualization, Writing - Review & Editing. **Vogelsang:** Conceptualization, Supervision.

[10]: **Fischbach:** Conceptualization, Methodology, Software, Validation, Investigation, Data Curation, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision. **Frattini:** Methodology, Software, Validation, Investigation, Data Curation, Writing - Review & Editing, Visualization. **Vogelsang:** Conceptualization, Supervision, Project administration. **Mendez:** Methodology, Writing - Review & Editing, Supervision. **Unterkalmsteiner:** Methodology, Writing - Review & Editing, Supervision. **Wehrle:** Data curation, Software. **Henao:** Software. **Yousefi:** Resources, Investigation. **Juricic:** Resources, Investigation. **Radduenz:** Resources, Investigation. **Wiecher:** Resources, Investigation.

Related Papers Not Included in This Thesis

The author of this dissertation has also co-authored the following papers that do not contribute to this thesis:

[11]: J. Fischbach, M. Junker, A. Vogelsang, and D. Freudenstein, "Automated generation of test models from semi-structured requirements," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pp. 263–269, 2019

[12]: P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 80–86, 2021

[13]: J. Bohn, J. Fischbach, M. Schmitt, H. Schütze, and A. Vogelsang, "Semi-automated labeling of requirement datasets for relation extraction," in *Proceedings of the 14th Workshop on Building and Using Comparable Corpora (BUCC 2021)*, (Online (Virtual Mode)), pp. 40–45, INCOMA Ltd., Sept. 2021

Luck Is What Happens When
Preparation Meets Opportunity.

Seneca

Who Stops Being Better
Stops Being Good.

Oliver Cromwell

Contents

List of Figures

List of Tables

List of Acronyms

AB Ada Boost 90

ABAQM Activity-Based Artifact Quality Model xviii, 11, 145–148, 150, 152, 163

AC Affirming the Consequent 40–43

ACC Acceptance Criteria 150, 152–155, 160, 162, 170, 188–190

ASD Agile Software Development 144, 145, 148, 161

BDD Behavior Driven Development 50

BERT Bidirectional Encoder Representations from Transformers iii, xvii, 10, 29–32, 87, 89–93, 123, 125, 126, 131, 132, 179, 185

BiLSTM Bidirectional Long Short-Term Memory 125, 126, 128, 129

BIS Business Information System 64, 183

BoW Bag-of-Words 90

BPE Byte-Pair Encoding 126, 132, 168

BPST Basic Path Sensitization Technique 33, 35, 169, 188

BUCs Business Use Cases 170

CDDL Collaboration Diagrams Description Language 50

CEG Cause-Effect-Graph xviii, 4, 11, 32–35, 37, 44, 52, 135, 166, 167, 169, 171–174, 178, 180, 181, 184, 187–190

CiRA Conditionals in Requirements Artifacts iii, xiv, xv, xviii, 10–13, 37, 47, 52, 87, 89, 93, 95, 129, 131–139, 141, 165–171, 176–181, 183, 184, 186, 187, 189, 191

CLS Classification Token 31, 90, 91, 126, 132, 168

CNL Controlled Natural Language 3, 49, 50

CSP Communicating Sequential Processes 50

DA Denying the Antecedent 40–42

DEP Dependency Tag 91–93

DistilBERT A Distilled Version of BERT 10, 29–32, 87, 123, 125, 126, 129

DL Deep Learning 29, 43, 46, 47, 92, 128

DT Decision Tree 90, 92

EARS Easy Approach to Requirements Syntax 38–40, 49

ELMo Embeddings from Language Model 30

ESA European Space Agency 77

GI German Informatics Society 78

GloVe Global Vectors for Word Representation 30

GPT-2 Generative Pre-trained Transformer 2 30, 31

IEEE Institute of Electrical and Electronic Engineers 22

ISO International Organization for Standardization 23, 144

ISTQB International Software Testing Qualifications Board 144

KNN K-Nearest Neighbor 90, 92

LR Logistic Regression 90

LSTM Long Short Term Memory 46

LTL Linear Temporal Logic xvii, 3, 20–22, 74, 82

ML Machine Learning 10, 11, 24, 43, 45–47, 87, 89, 90, 92, 93, 185, 186

MP Modus Ponens 40, 42, 43

MT Modus Tollens 40

NASA National Aeronautics and Space Administration 77

NB Naive Bayes 90

NL Unrestricted Natural Language iii, 4, 6, 8, 10, 11, 13, 17, 18, 20, 24–27, 29, 37, 39, 40, 43–47, 51, 52, 74, 89, 90, 95–97, 105, 120, 131–133, 143, 152, 164, 167, 185, 186

NLP Natural Language Processing 12, 13, 17, 24, 27, 29, 31, 45, 46, 48, 51, 52, 57, 87, 89–91, 115, 186, 189

NLPL Nordic Language Processing Laboratory 91

NP Noun Phrase 45

OOV Out-Of-Vocabulary 11, 30, 179

PAD Padding Token 30, 90, 126

PL Propositional Logic 20, 21, 78

POS Part-of-Speech 46, 91, 92, 98, 101, 115–118

PROTEUS PROtotype TExt Understanding System 44

PURE PUblic REquirements Data Set 57, 167

QA Quality Assurance 161, 162

RE Requirements Engineering iii, xv, 1–4, 7–10, 12, 13, 17, 19, 20, 22–24, 37, 40, 43, 47, 53, 55, 57, 58, 65, 73, 74, 77–79, 83, 85, 86, 90, 93, 107, 131, 135, 180, 183–186

RETNA REquirements to Testing in a NAtural way 51

RF Random Forest 90, 92

RNN Recursive Neural Networks 28

RNTN Recursive Neural Tensor Network xvii, xviii, 10–12, 27, 28, 87, 95, 106–108, 110, 112, 113, 115–117, 120, 121, 123, 186

RoBERTa A Robustly Optimized BERT Pretraining Approach iii, 10, 11, 29–32, 87, 95, 123, 125, 126, 129, 131, 132, 168, 186

RTCM Restricted Test Case Modeling 50

RUCM Restricted Use Case Modeling 50

SAFe Scaled Agile Framework 146

SCR Software Cost Reduction 50

SDLC Software Development Life Cycle 23

SDP Shortest Dependency Path 46

SE Software Engineering 22, 37, 38

SEP Seperator Token 31, 126

SOFL Structured Object-Oriented Formal Language 48

SST Subtree Syntax Tree 100, 102

SVM Support Vector Machines 90

TF-IDF Term Frequency–Inverse Document Frequency 30, 90

THEMAS The Energy Management System 167–169

TL Transfer Learning 10, 12, 29, 87, 89, 90, 185–187

UML Unified Modeling Language 50

US User Story 188, 190

VDM Vienna Development Method 48

Chapter 1

Introduction

This thesis focuses on conditionals in requirements artifacts and highlights **why** and **how** *Requirements Engineering* (RE) can benefit from the automated extraction of conditionals. In this chapter, we introduce conditional statements (short: conditionals) as the central topic of this thesis (see Section 1.1). We motivate why RE researchers and practitioners should deal with automated conditional extraction by outlining two use cases (see Section 1.2). In Section 1.3, we discuss the problems of implementing automated conditional extraction for these two use cases and formulate the problem statement of this thesis. Section 1.4 summarizes the major contributions of this thesis. Finally, we give an overview of the structure of the thesis in Section 1.5.

1.1 Subject of Interest: Conditionals in Natural Language

Conditional statements (e.g., *"If A and B, then C"*) are an integral part of everyday discourse because they allow us to express conditions and their consequences. A conditional statement is a grammatical structure consisting of two parts: an adverbial clause, often referred to as the *antecedent*, and a main clause, also known as the *consequent* [14]. The semantics of conditional statements has been intensively discussed in the last decades and has received notable attention in studies of various disciplines, e.g., in psychology [15], linguistics [16, 17, 18], and philosophy [19]. These studies demonstrate that conditionals are a complex linguistic pattern that can occur in a variety of forms (e.g., *explicit*/*implicit* conditionals, *marked*/*unmarked* conditionals). For example, the Conditional 1.1 (see below) is *marked* since the cue phrases *"if"* and *"then"* indicate the dependence between the *antecedent* and the *consequent*. The same relation can also be expressed as an *unmarked* conditional: *"A and B occur. C evaluates to true."* This conditional is semantically identical to its *marked* form, but it spans across two sentences and does not contain a cue phrase that signals the relationship of the *antecedent* and *consequent*. Both the Conditional 1.1 and Conditional 1.2 are *explicit*. Specifically, they contain information about the *antecedent* as well as the *consequent*. The Conditional 1.3 (see below) is *implicit*, because the *consequent* that C evaluates to true is not explicitly stated. Rather, the interaction of the *antecedent* and *consequent* is encoded in the predicate (i.e., *"leads to"* implies that A and B are the trigger for C to occur).

If $\underbrace{\text{A and B occur}}_{\text{antecedent}}$, then $\underbrace{\text{C evaluates to true}}_{\text{consequent}}$. (**Cond. 1.1:** marked and explicit)

$\underbrace{\text{A and B occur}}_{\text{antecedent}}$. $\underbrace{\text{C evaluates to true}}_{\text{consequent}}$. (**Cond. 1.2:** unmarked and explicit)

$\underbrace{\text{The occurrence of A and B}}_{\text{antecedent}}$ leads to $\underbrace{\text{C}}_{\text{consequent}}$. (**Cond. 1.3:** marked and implicit)

Conditionals do not only occur in different forms but can also be associated with different semantic meanings. The views on the notion of conditionals differ greatly among researchers, and no consensus has emerged so far. However, all researchers unanimously agree that, in general, the *antecedent* describes the circumstances under which the *consequent* occurs. Hence, the Conditional 1.4 (see below) specifies that the detection of an error ($event_1$) is the condition for showing an error message ($event_2$). In other words, if $event_1$ evaluates to true, then $event_2$ is also set to true.

If $\underbrace{\text{the system detects an error}}_{\text{antecedent}}$, $\underbrace{\text{an error message is shown}}_{\text{consequent}}$. (**Cond. 1.4:** marked and explicit)

Conditional statements thus contain logical knowledge about the dependency between certain events. Automatically extracting this embedded knowledge enables several analytical disciplines and is already used for *Question Answering* [20], *Event Prediction* [21, 22, 23], *Emergency Management* [24], *Medical Text Mining* [25, 26, 27], and *Information Retrieval* [28]. For example, Doan et al. [29] extracts conditionals from *Twitter* messages to identify factors causing stress, insomnia, and headache. Radinsky et al. [30] propose an approach capable of identifying conditionals in news articles to predict future events that can be caused by certain events.

As a result of the rich body of research on conditional extraction, it has become evident that conditional extraction can provide added value in many use cases. Interestingly, the potential of extracting conditionals from texts has not yet been leveraged for RE although conditionals are prevalent in requirements artifacts. In fact, we found that conditionals are often used in both traditional and agile requirements such as acceptance criteria to capture the intended behavior of a system [1, 4]. This thesis addresses this research gap and sheds light on **why** and especially **how** the extraction of conditionals can support RE.

1.2 Motivating Use Cases in the Context of Automated RE

RE represents a central step in the software engineering process and deals with the elicitation, analysis, documentation, validation, and management of requirements [31]. A requirement can be understood as *"(1) a need perceived by a stakeholder, (2) a capability or property that a system shall have, or (3) a documented representation of a need, capability, or property"* [32]. We refer with a requirements artifact or RE artifact to the documented representation of a requirement. Several studies [33, 34] found that poor execution of RE activities increases the risk of building a system that ultimately does not meet the demands of the stakeholders. Driven by this observation, multiple methods [35, 36, 37, 38, 39] have been developed to support RE practitioners during their activities. In fact, we have witnessed an increasing trend in the RE community to build tools for the (semi-)automation of RE tasks in recent years [40]. The demand for greater automation in RE stems primarily from the following two reasons:

Increasing Amount of Requirements Practitioners struggle to manually examine large collections of requirements in the development of modern systems. For example, practitioners fail to maintain traceability of requirements in their software repositories, especially in the case of large-scale agile system development [41]. Similar to Neto et al. [42], we found that practitioners face the challenge of creating the right acceptance tests for

their requirements and aligning them in case of changes [3]. Wagner et al. [43] reveal that project teams struggle to keep track of all requirements and their relationships, causing inconsistencies between requirements. Additionally, research has shown that practitioners encounter problems in controlling the quality of their RE artifacts, frequently resulting in project failures [44]. Hence, we need approaches that can process large volumes of requirements automatically in order to support practitioners in performing RE specific tasks.

Diverse Sources of Requirements A key objective of RE is the identification of desired system properties from the perspective of stakeholders. Traditionally, techniques such as interviews, brainstorming, and focus groups are used to gather requirements by interacting directly with relevant stakeholders [45, 46]. However, recent studies [47, 48] reveal that valuable information for software development teams exists in a variety of distributed sources that can not be accessed using conventional elicitation techniques. For example, user comments in social media and app stores can contain requirements-related information such as feature requests and problem reports [49]. The number of reviews grows rapidly every day making a manual analysis of user comments cumbersome (popular apps like *Facebook* receive around 4,000 reviews daily [50]). Therefore, we require approaches to automatically extract RE-related information from diverse sources and convey it to requirements engineers in an understandable and manageable form.

The focus of this thesis lies in the automatic extraction of conditionals embedded in requirements artifacts and is thus also related to the context of increasing automation in RE. Specifically, we argue that the automated extraction of conditionals can help to automate two RE tasks for which sufficient methods and tools are not yet available: *"acceptance test creation"* and *"dependency detection between requirements"*. In the following, we describe both use cases in detail.

Use Case 1: Automatic Acceptance Test Creation

The test case design is a very laborious activity that easily accounts for 40–70 % of the total effort in the testing process [51, 52]. This stems from the following challenges.

Challenge 1 Determining the right set of test cases that fully covers a requirement is a difficult task, especially for complex requirements. Our industrial survey on challenges in agile testing confirms the findings of other studies [42, 53, 54] that acceptance tests are often not systematically created, resulting in incomplete or excessive test cases (see Chapter 9). In the case of missing test cases, system defects are not (or only partially) detected. In contrast, excessive test cases lead to unnecessary testing efforts and increased test maintenance costs. Consequently, practitioners need to strike a balance between full test coverage and the number of required test cases.

Challenge 2 Currently, creating acceptance tests is a predominantly manual task due to insufficient tool support [55]. Existing approaches allow the derivation of test cases from semi-formal requirements [56, 57, 58] (e.g., expressed in *Controlled Natural Language* (CNL)) or formal requirements [59, 60] (e.g., expressed in *Linear Temporal Logic* (LTL)) but are not suitable to process informal requirements. However, studies [61, 62]

have shown that requirements are usually expressed in *Unrestricted Natural Language* (NL). To derive test cases from NL requirements, the specified system behavior and the combinatorics behind the requirement need to be understood. Specifically, we need to understand the embedded conditional statement to determine the correct combination of test cases that cover all positive and negative scenarios. This can be illustrated by the following Conditional 1.5:

$$\text{If} \underbrace{\text{the customer is older than 23 years}}_{\text{antecedent}_1} \underbrace{\text{and}}_{\wedge} \underbrace{\text{shows a valid driving license}}_{\text{antecedent}_2}, \underbrace{\text{the system does not charge an increased fee}}_{\text{consequent}_1}. \quad \textbf{(Cond. 1.5)}$$

The conditional contains two *antecedents "customer is older than 23 years"* and *"[customer] shows a valid driving license"* and one *consequent "the system does not charge an increased fee"*. The *antecedents* are connected by a conjunction indicating that the *consequent* depends on the occurrence of both *antecedents*. Currently, practitioners have to extract the conditional manually and determine the combinations of *antecedents* and *consequents* that need to be covered by test cases. This is not only cumbersome but also becomes increasingly error-prone with growing requirements complexity as the number of potential test cases increases by 2^n, where n is the number of *antecedents*. Therefore, test cases may be missed during manual creation or testing effort may be spent on irrelevant test cases.

We argue that automated conditional extraction combined with existing automatic test case derivation contributes to the alignment of RE and testing. Specifically, we show how extracted conditionals can be mapped to a *Cause-Effect-Graph* (CEG), from which test cases can be derived automatically (see Chapter 10). This leads to time savings as the expected system behavior is interpreted automatically, and to a better test coverage as the required test cases are determined by heuristics. Hence, we argue that automated conditional extraction represents an essential contribution towards a fully automated test case generation from NL requirements.

Figure 1.1 demonstrates the use case by means of an excerpt from the requirements specification of the open-source tool *Specmate* [63]. *Specmate* is a tool for model-based testing. To facilitate illustration, we have slightly altered the excerpt from its original form. It is apparent that the requirements specification contains a series of requirements (REQ) that describe the desired system behavior by conditionals. To automatically create suitable acceptance tests for these requirements, we must first extract their embedded conditional statements. Figure 1.1 exemplifies the extracted conditionals from REQ 1.2, REQ 2.2 and REQ 2.3. Subsequently, we create *Cause-Effect-Graphs* that reflect the relationship between the extracted *antecedents* and *consequents* and use existing methods [64] to derive a test suite from the CEG. To transfer the conditionals into a CEG, the conditionals have to be extracted in fine-grained form. In other words, we need to decompose *antecedents* and *consequents* into variables and conditions, respectively, and consider their combinatorics. The fine-grained extraction of conditionals is therefore necessary to bridge the gap between requirements and test cases (see green highlighting in Figure 1.1).

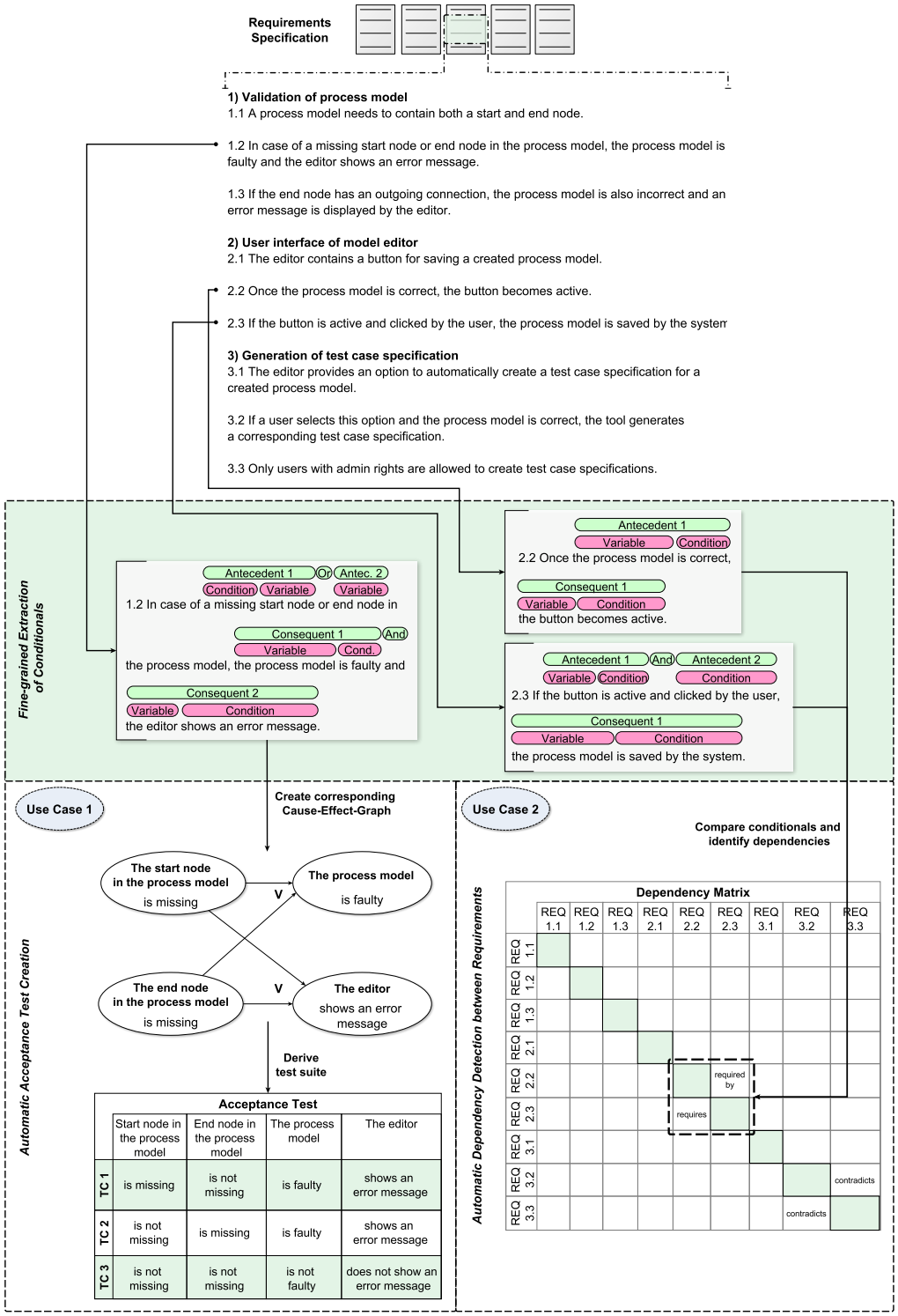


Figure 1.1: Use Cases of Conditional Extraction for Requirements Engineering. Left: Automatic Acceptance Test Creation (Use Case 1). Right: Automatic Dependency Detection Between Requirements (Use Case 2).

Use Case 2: Automatic Dependency Detection Between Requirements

As modern systems are becoming increasingly sophisticated, both the complexity of requirements and their quantity increase. Additionally, as requirements are primarily expressed by NL, it is difficult to keep an overview of all requirements and their relations [40]. Undetected redundancies and inconsistencies in the requirements may lead to faults in the system design [65]. Furthermore, knowledge about requirements relations is essential to understand the impact of a proposed software change on the overall system functionality [66]. To better understand the relations between requirements, different dependency models have been proposed describing relations on different abstraction levels. An integrated view is provided by Dahlstedt and Persson [67].

We argue that an automatic conditional extraction from requirements can help to compare the semantics by analyzing the different embedded conditional statements. As a result, relations between requirements can be identified automatically. We are convinced that conditional extraction contributes to the detection of four out of the seven dependency types described by Dahlstedt and Persson [67]:

1) Contradictory Requirements Two conditional statements c_1 and c_2 *contradict* each other, i.e. the content of two requirements conflict with each other. Such conflicts may lead to inconsistencies in the system design. Identifying these conflicts automatically by conditional extraction enables engineers to detect requirements that need to be rediscussed with the stakeholders to clarify the desired behavior. For example, the two conditionals below contradict each other, since c_1 allows the creation of nodes only by admins, while c_2 states that any user shall be able to create nodes.

c_1 : Only users with admin rights are allowed to create nodes.
c_2 : If a user selects this option, he/she can create a node in the editor.

2) Requirements That Require Others If the *consequent* of a conditional c_1 appears as an *antecedent* in another conditional c_2, the fulfillment of c_2 depends on c_1. In other words, the requirement that includes c_2 *requires* the requirement containing c_1. Indicating this kind of relation helps engineers to maintain requirements traceability. If c_1 is changed, the engineer is aware of which other requirements are affected. For example, c_2 below depends on c_1 because the button must be active first before the user can interact with it to save the process model.

c_1 : Once the process model is correct, the button becomes active.
c_2 : If the button is active and clicked by the user, the process mode is saved by the system.

3) Redundant Requirements The conditional statements are congruent, i.e. the two requirements describe the same system behavior and are therefore *redundant*. Engineers may remove one of the requirements to keep the requirements suite minimal. In the following example, c_1 and c_2 are equivalent, since they semantically describe the same

expected system behavior regardless of their different syntax (position of *antecedent* and *consequent* is reversed).

c_1 : In case of a missing end node, the editor shows an error message.
c_2 : An error message is displayed by the editor if an end node is not included in the process model.

equal to

4) Requirements Refinement A conditional c_1 that specifies a subset of the solution of another conditional c_2 indicates that c_1 *refines* c_2. In other words, the system behavior described in one requirement is defined more precisely in another. In the following example, c_2 is refined by c_1, which specifies that the existence of an outgoing connection from the end node is sufficient for the model to be incorrect. Therefore, the condition that an end node must be properly named is no longer required for a valid model.

c_1 : If an end node has an outgoing connection, the model is incorrect.
c_2 : If an end node is unnamed and has an outgoing connection, the model is incorrect.

refines

Figure 1.1 illustrates the idea behind the use case and shows how conditional extraction can help to identify dependencies in a requirements specification. Similar to Use Case 1, the first step is to identify and extract the conditionals contained in the requirements specification. Subsequently, we need to compare the semantics of the conditionals to detect potential dependencies. In the case of REQ 2.2. and REQ 2.3, we observe that the *consequent* of REQ 2.2. is adopted as an *antecedent* in REQ 2.3. Consequently, REQ 2.2 is *required by* REQ 2.3. Mapping these identified dependencies, for example in the form of a dependency matrix, proves useful to business analysts for maintaining an overview of all requirements.

Scope of This Thesis

This thesis constitutes the first work in the RE community that studies the potential of extracting conditionals from requirements. It is intended to stimulate further engagement of researchers and practitioners in the field of conditionals in RE artifacts. In essence, the dissertation presents fundamental research on the notion of conditionals in requirements as well as methods for their fine-grained extraction. We are convinced that our results can serve as foundation for automatic acceptance test creation (Use Case 1) and for the detection of dependencies between requirements (Use Case 2). In this thesis, however, we only present empirical evidence that automatic conditional extraction can contribute to the automatic generation of acceptance tests (see Part III). Hence, the scope of this thesis is limited to Use Case 1 (see left part of Figure 1.1). We encourage fellow researchers to apply our developed approaches also for the detection of dependencies between requirements or other use cases.

1.3 Problem Statement

We see two principal reasons why the potential of conditional extraction has not yet been exploited for RE:

Table 1.1: Existing Techniques for Conditional Extraction From Natural Language. A Detailed Overview of State of the Art in Automated Conditional Extraction Can Be Found in Section 3.2.

	State of the Art (Excerpt)	Example: If input A (variable) is true (condition) and input B (variable) is false (condition), the system (variable) shall show an error message (condition). (antecedent$_1$ = input A is true (∧) antecedent$_2$ = input B is false, consequent$_1$ = the system shall show an error message)	#
Word	Chang and Choi [68],	antecedent$_1$ = *true*, consequent$_1$ = *message*	c_1
	Rink et al. [69], Khoo et al. [26]	antecedent$_2$ = *false*, consequent$_1$ = *message*	c_2
Phrase	Dasgupta et al. [70],	antecedent$_1$ = *input A is true*, consequent$_1$ = *the system shall show an error message*	c_3
	Li et al. [71], Girju [20]	antecedent$_2$ = *input B is false*, consequent$_1$ = *the system shall show an error message*	c_4
Full	Our Scope	antecedent$_1$ = *input A is true* ∧ antecedent$_2$ = *input B is false*, consequent$_1$ = *the system shall show an error message*	c_5

Problem 1: Missing Understanding of the Notion of Conditionals in Requirements Artifacts

The extent, form, and complexity of conditional statements in requirements artifacts is not well understood. We lack empirical evidence on conditionals in traditional RE artifacts (e.g., requirements documents) and agile RE artifacts (e.g., acceptance criteria). Further, we do not know how authors of requirements formulate conditionals and in which complexity the conditionals usually occur: do they tend to specify only the dependency of a single *antecedent* and *consequent*, or do conditionals in RE artifacts include multiple interdependent events? We also do not know whether conditionals in RE artifacts typically occur in *marked* or *unmarked* form. This lack of knowledge hinders the development of approaches capable of extracting conditionals from requirements artifacts.

Even more importantly, we do not know how conditional statements are logically interpreted by RE practitioners. For example, we still lack insight into whether RE practitioners perceive *antecedents* only as sufficient or also necessary for the *consequents*. However, reliable knowledge about the logical interpretations of conditionals by RE practitioners is vital since conditionals need always be associated with a formal meaning to automatically process them. Otherwise, we choose a formalization that does not reflect how practitioners interpret conditional sentences, rendering downstream activities error-prone. That is, we would likely derive incomplete test cases or interpret dependencies between the requirements incorrectly.

Problem 2: Missing Tool-Supported Approach for Fine-Grained Extraction of Conditionals

The use cases described in Section 1.2 require conditionals to be extracted in fine-grained form. Specifically, we need to consider the combinatorics between *antecedents* and *consequents* and split them into more fine-granular text fragments (e.g., variable and condition), making the extracted conditionals suitable for automatic test case derivation and dependency detection. However, existing approaches are not able to extract conditional clauses from NL requirements in fine-grained form, which is illustrated by Table 1.1. Some approaches [68, 69, 72] extract *antecedents* and *consequents* only on word level (see extracted conditionals c_1 and c_2). Consequently, valuable information about the conditional statement is lost (e.g., the conditions of *"input A"*, *"input B"* and *"the system"* are ignored).

Recent approaches [70, 71] address this problem and identify conditionals on phrase level. Nevertheless, they only extract *antecedent-consequent* pairs, whereby the combinatorics between the *antecedents* and *consequents* gets lost during the extraction (see c_3 and c_4).

We need to extract the entire embedded conditional statement to make it usable for test case derivation and dependency detection between requirements (see c_5). Thus, we require a new conditional extraction approach to implement our described use cases. This approach should be accompanied by adequate tool support to be easily integrated into testing processes in practice.

Building on the two outlined problems, we formulate the following problem statement and address it within the scope of this thesis.

Problem Statement:

We need (1) a better understanding of the notion of conditionals in requirements artifacts and (2) a comprehensive method and tool support to extract conditionals in fine-grained form.

1.4 Contributions

This dissertation addresses both problems and makes six contributions. According to the taxonomy introduced by Stol and Fitzgerald [73], we present both *knowledge-seeking* as well as *solution-seeking* studies. Our *knowledge-seeking* studies are devoted to the research question of **why** conditionals should be extracted from RE artifacts. In the context of our *solution-seeking* studies, we investigate the research question of **how** to extract conditionals from RE artifacts automatically. Hence, the outcome of our *knowledge-seeking* research are mainly empirical findings, while our *solution-seeking* research aims at the development of artifacts such as algorithms and tools. In the following, we describe our contributions and indicate whether they belong to the *knowledge-seeking* (🔍) or *solution-seeking* (⚙) part of the thesis.

Contribution 1: Empirical Study on Prevalence, Form, and Complexity of Conditionals in RE Artifacts (🔍)

We report on an exploratory case study where we analyze the prevalence, form, and complexity of conditionals in requirements based on 14,983 sentences emerging from 53 requirement documents. These documents originate from 18 different domains and were written between 2004 and 2019, making our analysis not restricted to a single year or domain, but rather allows for a comprehensive and generalizable view on conditionals in requirements. Our case study corroborates, among other things, that:

Published at:

REFSQ [4]
18 pages
Best Full Paper 🏆

REJ [7]
38 pages
Full Paper

- Conditional statements are a major linguistic element of requirement artifacts as they occur in about 28 % of the analyzed sentences. Therefore, conditionals matter in RE, which motivates the necessity of an effective and reliable approach for the automatic extraction of conditional statements.

- The majority of conditionals occur in *marked* form and contain one or more cue phrases to indicate the dependence between certain events. Only around 15 % of the investigated sentences were categorized as *unmarked*.
- The complexity of conditionals is confined since they usually consist of a single *antecedent* and *consequent* relationship in all observed, eligible domains. Two to three *antecedents* and two *consequents* were predominantly prevalent in the case of complex conditional statements. More than three *antecedents* were rare.

Contribution 2: Empirical Study on the Logical Interpretation of Conditionals by RE Practitioners (🔍)

Published at:
PROFES [5]
16 pages
Full Paper

We report on a descriptive survey with 104 RE practitioners in which we ask how they interpret 12 different conditional clauses. We map their interpretations to logical formulas written in *Propositional (Temporal) Logic* to provide empirical evidence for whether a common formal interpretation of conditionals in requirements exists. Key insights include but are not limited to:

- Conditionals in requirements are ambiguous. Practitioners disagreed on whether an *antecedent* is only sufficient or also necessary for a *consequent*.
- We observed a statistically significant relationship between the interpretation and certain context factors of practitioners (e.g., experience in RE, the way how a practitioner interacts with requirements, and the presence of domain knowledge). Interestingly, domain knowledge does not promote a consistent interpretation of conditionals.
- The choice of certain cue phrases has an impact on the degree of ambiguity (e.g., *"while"* was less ambiguous than *"if"* or *"when"* w.r.t. temporal relationship between *antecedent* and *consequent*).

Contribution 3: CiRA - An Approach for the Automatic Extraction of Conditionals From RE Artifacts (⚙)

Published at:
ICST [1]
11 pages
Full Paper
REFSQ [4]
18 pages
Best Full Paper
AIRE [6]
10 pages
Full Paper
Under Review at:
JSS [10]
17 pages
Full Paper

We present our tool-supported approach named CiRA (Conditionals in Requirements Artifacts), which consists of two steps: It first detects whether an NL requirement contains a conditional. Second, CiRA extracts the conditional in fine-grained form. We implement different methods for both steps and compare them with each other. Specifically, we investigate the performance of rule-based approaches, *Machine Learning* (ML) approaches (e.g., *Random Forest*, *Support Vector Machines*), and *Transfer Learning* (TL) approaches (e.g., BERT) in detecting conditionals. For the fine-grained extraction of conditionals, we implement three approaches and compare their performance with each other: *Dependency Parsing*, *Recursive Neural Tensor Network* (RNTN), and TL approaches (e.g., RoBERTa, DistilBERT). Our experiments show that:

- According to our evaluation on a real-world data set of 8,430 sentences, syntactically enriched BERT embeddings combined with a softmax classifier outperform other methods in detecting conditional statements (macro-F_1 score: 82 %). However, the application of TL does not result in a large performance boost over conventional

ML methods (gain of only 4 % in macro-F_1 compared to the best ML method). The rule-based approach is not able to distinguish between sentences that contain conditionals (F_1 score: 66 %) and sentences that do not (F_1 score: 64 %).

- The *Dependency Parsing*-based approach fails to extract conditionals in the case of grammatical errors in a NL sentence. Our RNTN-based approach performs better in handling grammatical errors, yet struggles to understand the semantics of *Out-Of-Vocabulary* (OOV) words posing a threat to its applicability in practice. Contrary, we found that a sigmoid classifier built on RoBERTa embeddings is more robust and best suited to extract conditionals in fine-grained form. It achieves a macro-F_1 score of 86 % when evaluated on a real-world data set of 1,946 sentences.

Contribution 4: Empirical Study on Challenges in the Quality Control of Agile Test Artifacts (Q)

Published at:

ESEM [3]
10 pages
Best Industry Paper

In Section 1.2, we claim that practitioners still have to manually derive acceptance tests from requirements due to missing tool support and that they struggle to identify the minimal set of required test cases. We present an industrial survey with 18 practitioners from 12 companies operating in seven different domains that provides supporting evidence for this claim. In our survey, we analyze *how* agile test artifacts are designed in practice and *which* properties make them useful for quality assurance. We draw the following main conclusions:

- We identified nine challenges that practitioners face when using the test artifacts acceptance criterion, acceptance test, feature, test documentation, test data, and unit tests. Interestingly, we observed mostly challenges regarding language and traceability, which are well-known to occur in non-agile projects.
- One of the most frequently stated problems concerned the lack of adequate acceptance tests. We found that acceptance tests are often not systematically created, resulting in incomplete or excessive test cases.
- We present an *Activity-Based Artifact Quality Model* (ABAQM) of 16 quality factors for agile test artifacts, serving as a foundation for systematic quality control in practice.

Contribution 5: Empirical Study on Utilizing CiRA for Automatic Acceptance Test Creation (Q)

Under Review at:

JSS [10]
17 pages
Full Paper

We report on a case study with three industry partners and demonstrate how CiRA can help to create acceptance tests automatically (Use Case 1). To this end, we utilize CiRA to extract conditionals in fine-grained form and map them to a CEG, from which the minimal set of required test cases can be derived automatically. We empirically evaluate our approach on real-world data provided by *Allianz Deutschland AG* (insurance), *Ericsson* (telecommunication), and *Leopold Kostal GmbH & Co. KG* (automotive).

- Across all case companies, our approach automatically created 71.8 % of the 578 manually created test cases. Our approach was further able to identify 136 test cases that were missed in manual test design. In fact, 58.8 % of these exclusively

automatically generated test cases are indeed relevant and should be included in the acceptance test. We conclude that our approach is able to automatically create a significant amount of relevant (known and new) test cases.

- In our setting, we observed four main reasons of deviations in test cases: incomplete requirements, incorrect combinatorics, infeasible test cases, and errors of our approach. We found that incomplete requirements are the main reason for test cases that could not be created automatically.

Contribution 6: Replication Packages and Online Demos (🔍 and ⚙)

Open Science ensures reliable research and increases the efficiency within the research community by providing greater access to scientific inputs and outputs. We join this initiative and make our code, annotated data sets, and trained models publicly available. We encourage fellow researchers and practitioners to use our research artifacts for their studies. We publish the following artifacts bundled with this thesis:

- A gold standard corpus of 212,186 extracted sentences from 463 publicly available requirement documents. We encourage fellow researchers to use the gold standard corpus for their RE-relevant *Natural Language Processing* (NLP) tasks. The corpus can be found at `https://figshare.com/s/725309c06b9dc82aa4a1`.

- A data set of 14,983 requirements sentences annotated regarding the prevalence, form, and complexity of conditional statements. This data set is the main research output of the first empirical study included in this thesis (see Chapter 4). We use this corpus to train our detection approaches (see Chapter 6). The annotated data set as well as the code of our detection approaches can be found at `https://github.com/fischJan/CiRA`.

- A data set containing (1) the survey protocol and (2) the survey responses of our study on how practitioners interpret conditional requirements (see Chapter 5). The replication package can be found at `https://doi.org/10.5281/zenodo.5070235`.

- The *Conditional Treebank*, which is the first corpus of fully labeled binary parse trees representing the composition of 1,571 requirements that contain conditionals. We use this corpus to train our extraction approach based on an RNTN (see Section 7.2). Our code for the implementation of the RNTN and the corpus are available at `https://github.com/springto/Fine-Grained-Causality-Extraction-From-NL-Requirements`.

- A data set of 1,946 requirements containing conditional statements annotated in a fine-grained manner. We use this data set to train our extraction approach based on TL models (see Section 7.3). Our code, the training corpus, and all trained TL models can be found at `https://doi.org/10.5281/zenodo.5550387`

To enable interested researchers and practitioners to easily interact with our extraction approach, we implemented an online demo. `CiRA` is available at `http://www.cira.bth.se/demo/`.

1.5 Outline and Instructions for Readers

Structure of This Thesis Figure 1.2 illustrates the structure of this thesis, its main contributions, and created research artifacts. In the present chapter, we motivated why researchers and practitioners should deal with conditional extraction by outlining two use cases. The need for the realization of these use cases is empirically grounded by our industrial survey on challenges in the quality control of agile artifacts (see Chapter 9) and other related studies [42, 53, 54]. Further, we discussed the problems of implementing conditional extraction for these two use cases and formulated the problem statement of our thesis. In Chapter 2, we present the fundamentals that are needed to comprehend the content of this thesis. In particular, we define the used terminology of this thesis and provide an overview of relevant NLP methods that we utilize to extract conditionals in fine-grained form. In Chapter 3, we discuss the current state of the art structured along with the contributions of this thesis considering empirical work on the notion of conditionals in NL, approaches for automated conditional extraction as well as approaches for automated test case derivation. To systematically address our problem statement, we organize the remainder of the thesis into three parts:

In Part I, we address the first problem, namely *"the missing understanding of conditionals in RE artifacts"*. To this end, we present empirical results on both the prevalence, form and complexity as well as on the logical interpretation of conditionals in requirements artifacts (see Chapter 4 and Chapter 5). Our findings serve as foundation for the development of an approach for automated conditional extraction from RE artifacts.

In Part II, we introduce and compare different methods for the detection and extraction of conditionals from NL requirements considering the results of Part I. Our investigated methods for the detection of conditionals are discussed in Chapter 6. We present our implemented conditional extraction approaches in Chapter 7. We combine the best performing methods for both steps and present `CiRA` as a solution for the second problem (*"the fine-grained conditional extraction"*) in Chapter 8.

In Part III, we provide empirical evidence demonstrating that the lack of adequate acceptance tests is one of the major problems in agile testing (see Chapter 9). We address this problem by applying `CiRA` in an industrial case study and showing how conditional extraction can contribute to the automatic generation of acceptance tests (see Chapter 10). We thus empirically prove that conditional extraction can indeed facilitate the realization of Use Case 1 (*"acceptance test creation"*) as described in Section 1.2.

Chapter 11 summarizes this thesis by describing its contributions, limitations and directions for future work.

Instructions for Readers Readers of this thesis may be interested only in certain parts of our work and therefore do not necessarily want to read the thesis from cover to cover. Rather, they want to focus on the chapters that are most relevant to them. This reading style is reasonable since readers may originate from different disciplines and possess varying levels of background knowledge. However, we stress that the contents of

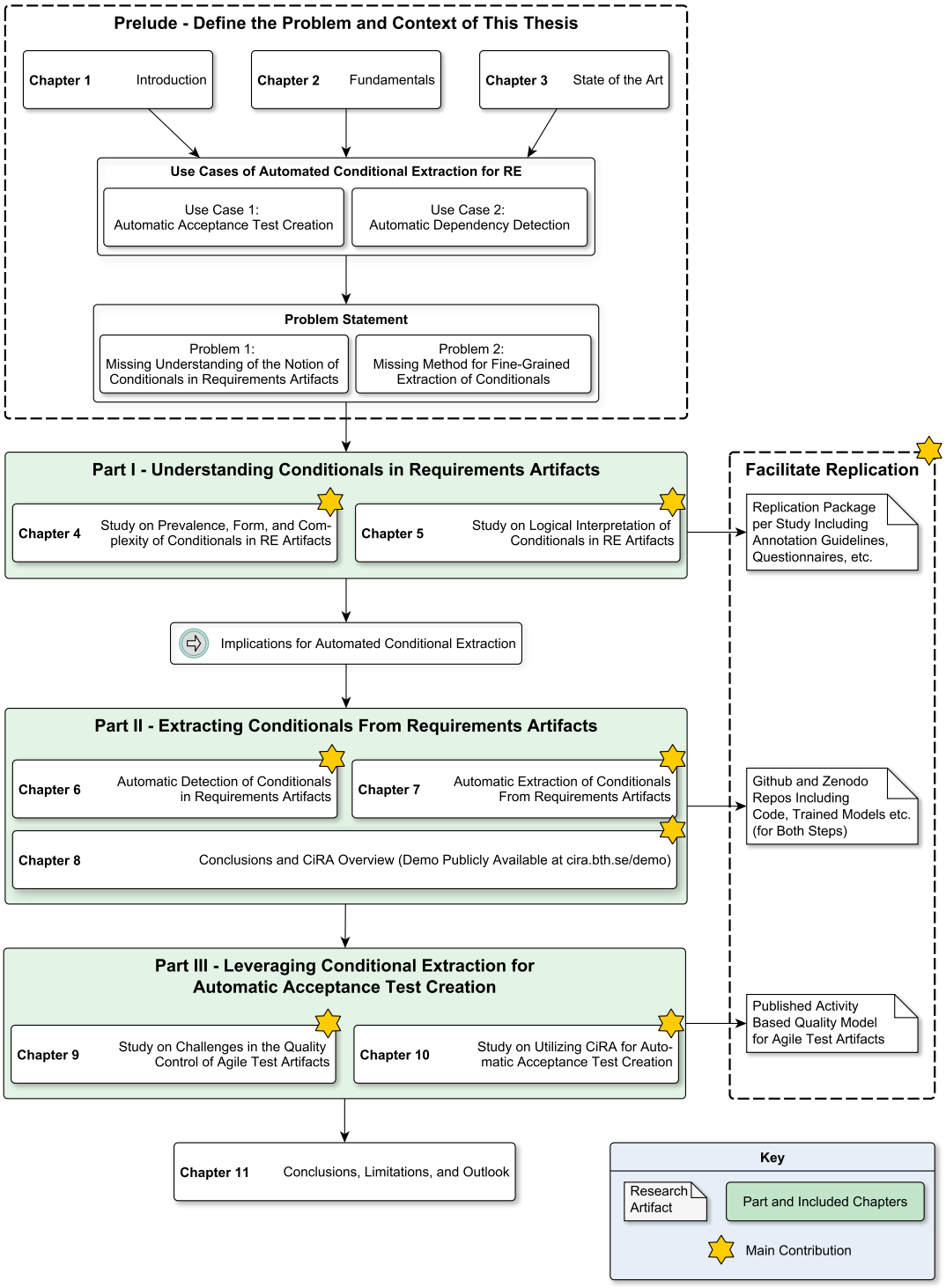


Figure 1.2: Overview of the Structure of the Thesis and Its Main Contributions.

the chapters are strongly interwoven. This requires readers to first understand certain chapters before they can follow the others. To highlight the connection between the individual parts and chapters, we included preambles to chapters and parts helping the reader to understand the following aspects:

Common Thread Describes how the chapter or part fits into the common thread of the thesis. This ensures that the reader is always aware of the link of a certain chapter or part to its predecessors. For example, the implementation of conditional extraction approaches in Part II is driven by the results of Part I.

Contribution Summarizes the key findings of a chapter and highlights how it contributes towards addressing our problem statement. In essence, we formulate an abstract of each chapter.

Preliminaries Points out to theoretical basics that are necessary for the understanding of a certain part. For example, to understand Part II the reader needs to possess a solid understanding of specific *Natural Language Processing* methods that are described in Section 2.3.

Related Publications Indicates on which publications a specific chapter is based.

We encourage readers to consider the information provided in the preambles as a guide to the thesis. It should be noted that not every preamble includes all of these four aspects. For example, the *"related publications"* information is provided at the chapter level rather than the part level to clearly map the body of this thesis to our corresponding publications.

Conventions We rely on American English and use the singular *they*, including the related forms *them*, *their*, and *themselves*, as the gender-neutral singular pronoun throughout this dissertation. To highlight important text fragments such as summaries or key-takeaways, we use boxes of the following style:

A Title:

Some content.

We use *"italic font in quotes"* to denote direct citations of authors or to highlight certain expressions. Text in *italic font* refers to a proper name or is used to emphasize a notable word. We use green-ish font to indicate references to parts, chapters, sections, figures, tables, and a full description of certain acronyms. This enables the reader, for example, to look up the meaning of a particular acronym at any point in the thesis. We use framed expressions to accentuate key concepts that are repeatedly mentioned throughout the work such as the names of use cases and annotation labels.

In case we use company names, such as *Allianz Deutschland AG*, *Ericsson*, or *Google*, as well as product names, such as *Github*, *Jira*, or *spacy*, it is understood that these names are registered trademarks.

Chapter 2

Fundamentals

This chapter introduces the theoretical background of this thesis. In Section 2.1, we define the term *conditional statement* as the central subject of investigation. Section 2.2 introduces the foundations of RE and explains how automated conditional extraction integrates into the end-to-end software development process. To extract conditionals from NL requirements, we use a variety of different NLP methods. Section 2.3 describes these methods to the extent necessary in the context of this thesis. To facilitate the realization of Use Case 1, we combine our conditional extraction approach with *Cause-Effect-Graphing* which can be used to derive acceptance tests automatically. Section 2.4 presents the idea behind *Cause-Effect-Graphing* by means of a running example. This chapter is partly based on previous publications [1, 4, 5, 6].

2.1 Terminology: What Is a Conditional?

This section introduces the term conditional and discusses several forms in which conditionals can occur (see Section 2.1.1). We also describe that conditionals can be interpreted in different ways and provide an overview of formal languages that can be used to formalize these interpretations (see Section 2.1.2).

2.1.1 Notion of Conditionals

A conditional statement (short: conditional) represents a grammatical structure that consists of two parts: an adverbial clause, often referred to as the *conditional clause*, *antecedent*, or *protasis*, and a main clause, also known as the *consequent* or *apodosis* [14]. The semantics of conditionals in NL has been extensively studied by a number of different disciplines, such as psychology [15], linguistics [16, 17, 18], and philosophy [19]. The views on the notion of conditionals differ greatly among the studies, and no consensus has emerged so far. However, the authors agree that, in general, the *antecedent* describes the conditions under which the *consequent* evaluates to true. Hence, a conditional specifies that the *consequent* is dependent on the occurrence of the *antecedent* (i.e., a conditional defines a contingency relation). In the following example, the window only moves down as long as the button is pressed.

While $\underbrace{\text{the button is in movement down position}}_{\text{antecedent}}$, $\underbrace{\text{the window moves down}}_{\text{consequent}}$. (**Cond. 2.1**)

Both the *antecedent* and *consequent* can be understood as an event, which is commonly defined as *"any situation (including a process or state) that happens or occurs either instantaneously (punctual) or during a period of time (durative)"* [74]. In the case of conditional 2.1 (see above), the *antecedent* and *consequent* are of durative nature. The

type of dependency between an *antecedent* (e_1) and *consequent* (e_2) can take one of three different forms [75]: a *causing*, *enabling*, or *preventing* relationship.

- e_1 **causes** e_2: If e_1 occurs, e_2 also occurs. This can be illustrated by the following conditional c_1: *"After the user enters a wrong password, a warning window shall be shown."* In this case, the wrong input is the trigger to display the warning window.
- e_1 **enables** e_2: If e_1 does not occur, e_2 does not occur either (i.e., e_2 is not enabled). Let us assume c_2: *"As long as you are a student, you are allowed to use the sport facilities of the university."* Only the student status enables to do sports on campus.
- e_1 **prevents** e_2: If e_1 occurs, e_2 does not occur. This can be illustrated by c_3: *"Data redundancy is required to prevent a single failure from causing the loss of collected data."* There will be no data loss due to data redundancy.

Forms of Conditionals The form in which conditionals can be expressed has three further characteristics [76]: *marked* and *unmarked* conditionals, *explicit* and *implicit* conditionals, and *ambiguous* and *non-ambiguous* regarding its cue phrases, a linguistic concept commonly used when dealing with conditionals in NL [77, 68]. A cue phrase is defined as *"a word, a phrase, or a word pattern, which connects one event to the other with some relation"* [68] and represents therefore a lexical indicator for the dependence of *antecedent* and *consequent*.

- **Marked and unmarked:** A conditional is *marked* if a certain cue phrase indicates the dependence between *antecedent* and *consequent*. The conditional *"If the user presses the button, a window appears"* is *marked* by the cue phrase *"if"*, while *"The user has no admin rights. He cannot open the folder"* is *unmarked*.
- **Explicit and implicit:** An *explicit* conditional contains information about both the *antecedent* and *consequent*. The conditional *"In case of an error, the system prints an error message to the console"* is *explicit* since it contains both the *antecedent* (error) and *consequent* (error message). *"A parent process kills a child process"* is *implicit* because the *consequent* that the child process is terminated is not explicitly stated. Rather, the interaction of *antecedent* and *consequent* is encoded in the predicate (i.e., *"kills"* implies that the child process is stopped by the parent process). The presence of *implicit* conditionals is therefore dependent on the use of specific verbs [78]. Garvey and Caramazza [79] provide an overview of verbs that, in their words, *"stem from a class exhibiting implicit causality"*. Implicitly conditionals are particularly hard to process and might be a potential source of ambiguity in requirements due to their obscuring nature.
- **Ambiguous and non-ambiguous cue phrases:** Due to the specificity of most cue phrases in *marked* conditionals, it seems feasible to deduce the classification of a sentence as containing a conditional based on the occurrence of certain cue phrases. However, certain cue phrases (e.g., *"since"*) indicate conditionals, but also occur in other contexts (e.g., denoting time durations). Such cue phrases are called *ambiguous*, while cue phrases (e.g., *"because"*) that predominantly indicate conditionals are called *non-ambiguous*.

Complexity of Conditionals Our previous explanations refer to the simplest case where the conditional consists of a single *antecedent* and *consequent*. With increasing system complexity, however, the expected system behavior is described by multiple *antecedents* and *consequents* that are connected to each other. They are linked either by conjunctions ($e_1 \wedge \dots$) or disjunctions ($e_1 \vee \dots$) or a combination of both which increases the complexity of the conditional statement. Furthermore, conditionals can not only be contained in a single sentence, but also span over multiple sentences, which is a significant challenge for conditional extraction. Additionally, the complexity increases when several conditional statements are linked together, i.e. if the *consequent* of a conditional c_1 represents an *antecedent* in another relation c_2. We define such conditionals, where c_2 is dependent on c_1, as *event chains*.

Conditionals vs. Causation In everyday language, conditionals like *"If A, then B"* are often conceived as causal relations. Specifically, *antecedents* are usually understood as causes (see *"A"*) and *consequents* as effects (*"B"*). Hence, the terms conditionals and causation are often used interchangeably, although they represent completely different concepts. A conditional is a linguistic pattern that describes a dependence between an *antecedent* and a *consequent*. In other words, the *antecedent* and *consequent* are associated [80]. Causation is more specific and represents a distinctive form of an association. To turn an association into a causal relationship, three constraints must be satisfied [81, 82]:

- **Constraint 1:** The causing event (cause) must be both *sufficient* and *necessary* for the caused event (effect) [83]. Consequently, the connection between cause and effect is counterfactual: If the cause did not occur, then the effect could not have occurred either [84].
- **Constraint 2:** The effect occurs either simultaneously with or after the cause [85].
- **Constraint 3:** The cause must occur independently (i.e., there is no confounder that influences the cause and effect and incorrectly implies causation) [86].

One sees immediately that a conditional describes any relationship between an *antecedent* and a *consequent*, while causation is a specific type of relationship for which a number of constraints must be met. Hence, we can conclude that a conditional does not imply causation: conditionals can arise in presence (i.e., *"A"* causes *"B"*) or absence (i.e., *"A"* and *"B"* have a common cause) of a causal relationship [87]. It is therefore misleading to always interpret *antecedents* as causes and *consequents* as effects when analyzing the meaning of a conditional. We explicitly do not deal with causation in the context of this thesis, but rather more fundamentally with conditionals in RE artifacts.

However, we argue that causation is often the main focus when formulating conditionals in RE artifacts [2]. As a requirements author, I want to formulate the system behavior precisely by defining an *antecedent* as both the sufficient and necessary reason for the occurrence of a *consequent* (see Constraint 1). In other words, if *"A"* occurs, *"B"* should also occur and if *"A"* does not evaluate to true, then *"B"* should also not occur. In practice, it is common to formulate several requirements that describe the same *consequent* (e.g., *"When C occurs, then B"*). In this context, we assume that each requirement describes a separate case in which the *consequent* should occur and link their *antecedents* with disjunctions (i.e., $A \vee C \Leftrightarrow B$).

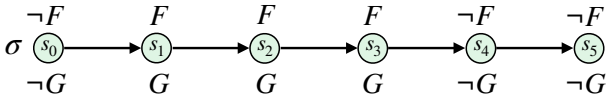
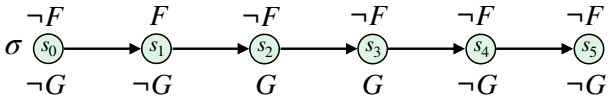
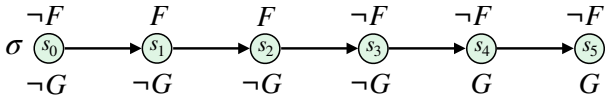
LTL Formula	Intended System Behavior (Example)
$\square(F \implies G)$	
$\square(F \implies \bigcirc G)$	
$\square(F \implies \Diamond G)$	

Figure 2.1: Overview of LTL Operators.

2.1.2 Logical Interpretation of Conditionals

As indicated in Section 2.1.1, there are many ways to express conditional statements in NL. Hence, the syntax can vary greatly among conditionals. Multiple studies [88, 89, 90] demonstrate that conditionals can also be associated with different semantic meanings, which makes them a source of ambiguity. In this thesis, we investigate the logical interpretations of conditionals by RE practitioners with respect to two dimensions: *Necessity* and *Temporality*. This section demarcates both dimensions and introduces suitable formal languages that can be used to formalize the interpretations appropriately. We use the following conditional as a running example: *"If the system detects an error (e_1), an error message shall be shown (e_2)".*

Necessity The relationship between an *antecedent* and *consequent* can be interpreted logically in two different ways. First, by means of an implication as $e_1 \Rightarrow e_2$, in which e_1 is a *sufficient* condition for e_2. Interpreting the running example as an implication requires the system to display an error message if e_1 is true. However, it is not specified what the system should do if e_1 is false. The implication allows both the occurrence of e_2 and its absence if e_1 is false. In contrast, the relationship of *antecedent* and *consequent* can also be understood as a logical equivalence, where e_1 is both a *sufficient* and *necessary* condition for e_2 (i.e., $e_1 \Leftrightarrow e_2$). Interpreting the running example as an equivalence requires the system to display an error message *if and only if* it detects an error. Consequently, if e_1 is false, then e_2 should also be false. The interpretation of conditionals as an implication or equivalence significantly influences further development activities. For example, a test designer who interprets conditionals rather as implication than equivalence might only add positive test cases to a test suite. This may lead to a misalignment of tests and requirements in case the business analyst actually intended to express an equivalence.

Temporality The temporal relation between an *antecedent* and *consequent* can be interpreted in three different ways: (1) the *consequent* occurs simultaneous with the *antecedent*, (2) the *consequent* occurs immediately after the *antecedent*, and (3) the *consequent* occurs at some indefinite point after the *antecedent*. *Propositional Logic* (PL)

does not consider temporal ordering of events and is therefore not expressive enough to model temporal relationships. In contrast, we require temporal logic (e.g., LTL), which considers temporal ordering by defining the behavior σ of a system as an infinite sequence of states $\langle s_0, \dots \rangle$, where s_n is a state of the system at *"time"* n [91]. Accordingly, requirements are understood as constraints on σ. The desired system behavior is defined as an LTL formula F, where next to the usual PL operators also temporal operators like $\Box$ (*always*), $\Diamond$ (*eventually*), and $\bigcirc$ (*next state*) are used. Since we will use these temporal operators in the course of the thesis, we will present them here in more detail.

To understand the LTL formulas, we assign a semantic meaning $[\![F]\!]$ to each syntactic object F. Formally, $[\![F]\!]$ is a boolean-valued function on σ. According to Lamport [91], $\sigma[\![F]\!]$ denotes the boolean value that formula F assigns to behavior σ, and that σ satisfies F if and only if $\sigma[\![F]\!]$ equals true (i.e., the system satisfies requirement F). We define $[\![\Box F]\!]$, $[\![\Diamond F]\!]$ and $[\![\bigcirc F]\!]$ in terms of $[\![F]\!]$ (see equations below). The expression $\langle s_0, \dots \rangle[\![F]\!]$ asserts that F is true at *"time"* 0 of the behavior, while $\langle s_n, \dots \rangle[\![F]\!]$ asserts that F is true at *"time"* n.

$$\forall n \in \mathbb{N} : \langle s_n, \dots \rangle[\![\Box F]\!] \Rightarrow \forall m \in \mathbb{N}, m \geq n, \langle s_m, \dots \rangle[\![F]\!] \tag{2.1}$$

$$\forall n \in \mathbb{N} : \langle s_n, \dots \rangle[\![\Diamond F]\!] \Rightarrow \exists m \in \mathbb{N}, m \geq n, \langle s_m, \dots \rangle[\![F]\!] \tag{2.2}$$

$$\forall n \in \mathbb{N} : \langle s_n, \dots \rangle[\![\bigcirc F]\!] \Rightarrow \langle s_{n+1}, \dots \rangle[\![F]\!] \tag{2.3}$$

Equation 2.1 asserts that F is true in all states of behavior σ. More specifically, $\Box F$ asserts that F is *always* true (now and forever). The temporal operator $\Diamond$ can be interpreted as *"it is not the case that F is always false"* (i.e., $\neg \Box \neg F$) [91]. According to Equation 2.2, a behavior σ satisfies $\Diamond F$ if and only if F is true at some state of σ. In other words, $\Diamond F$ asserts that F is *eventually* true (now or sometime in the future). According to Equation 2.3, $\bigcirc F$ asserts that F is true at the *next state* of behavior σ. In contrast to $\Diamond F$, $\bigcirc F$ requires that this state is not an arbitrary state of behavior σ, but rather the direct successor of state n. In conclusion, LTL can be used to incorporate temporal ordering into an implication $(F \Rightarrow G)$ in three ways:

- G occurs simultaneous with F:
  $\Box(F \Rightarrow G)$, which can be interpreted as *"any time F is true, G is also true"*.
- G occurs immediately after F:
  $\Box(F \Rightarrow \bigcirc G)$, which can be interpreted as *"G occurs after F terminated"*.
- G occurs at some indefinite point after F:
  $\Box(F \Rightarrow \Diamond G)$, which can be interpreted as *"any time F is true, G is also true or at a later state"*.

We illustrate the intended system behavior expressed by these formulas in Figure 2.1.

Formalization Matrix To distinguish the logical interpretations and their formalization, we constructed a formalization matrix (see Figure 2.2). It defines a conditional statement of F and G along the two dimensions (*Necessity* and *Temporality*), each divided on a

Necessity	Temporality			
	Temporal Ordering Relevant			VI. Temporal Ordering Not Relevant
	III. G is caused *during* F is true	IV. G will be caused in the *next state*	V. G will be caused *eventually*	
I. F is only *sufficient*	$\square(F \implies G)$	$\square(F \implies \bigcirc G)$	$\square(F \implies \Diamond G)$	$F \implies G$
II. F is also *necessary*	$\square(F \iff G)$	$\square(F \iff \bigcirc G)$	$\Diamond G \implies (\neg G \mathcal{U} F)$	$F \iff G$

Figure 2.2: Formalization Matrix Defining a Conditional of F and G Along the Two Dimensions *Necessity* and *Temporality*.

nominal scale. Specifically, the dimension *Necessity* has two levels: F is only *sufficient* or also *necessary* for G. The dimension *Temporality* has four levels: *during*, *next state*, *eventually*, and temporal ordering is not relevant. Each 2-tuple of characteristics can be mapped to an entry in the formalization matrix. For example, the LTL formula $\square(F \Rightarrow \bigcirc G)$ formalizes a conditional statement, in which F is only *sufficient* and G occurs in the *next state*.

To define F as both *sufficient* and *necessary* for G, we replace the implication by an equivalence and rephrase the LTL formula as follows: $\square(F \Leftrightarrow \bigcirc G)$. However, the equivalence operator is not adequate in cases where G will be caused *eventually*. Specifically, the formula $\square(F \Leftrightarrow \Diamond G)$ would define that as soon as F evaluates to false, G is locked permanently. We argue that this formula does not represent the behavior we want to express, since there may also be scenarios in which F is initially false, but turns true at a later state and leads to the occurrence of G. Therefore, we want to specify that as soon as G occurs, F must have occurred concurrently or at a previous state (i.e., F is a *necessary* condition for an occurrence of G). To this end, we build on the *precedence relation* introduced by Dwyer et al. [92]: $\Diamond G \Rightarrow (\neg G \,\mathcal{U}\, (F \wedge \neg G))$. The core element of the *precedence relation* is the until $\mathcal{U}$ operator. Literally, the *precedence relation* can be interpreted as *"If G occurs eventually, then G has been false until the state in which F occurs without G occurring concurrently."* Hence, in its original form, the *precedence relation* defines F as a *necessary* pre-condition of G. Since the *eventually* operator allows that G and F occur simultaneously, we adapt the *precedence relation* as follows: $\Diamond G \Rightarrow (\neg G \,\mathcal{U}\, F)$.

2.2 Role of Conditional Extraction in Requirements Engineering

In this section, we introduce the foundations of RE and describe its role in *Software Engineering* (SE). Further, we explain in which RE activities conditionals are involved and how automated conditional extraction integrates into the end-to-end software development process. Hence, the aim of this section is to familiarize the reader with the domain context of this thesis.

Requirements Engineering Is a Branch of Software Engineering According to the IEEE collection of standards, SE can be understood as *"the application of a systematic,*

disciplined, quantifiable approach to the development, operation, and maintenance of software, i.e. the application of engineering to software [...]" [93]. This definition emphasizes that SE represents an engineering approach that aims to develop software by performing well-defined and ordered activities [94, 95]. These activities and their interdependencies are described in a *Software Development Life Cycle* (SDLC). Over the last decades, a series of SDLC models have been developed, each proposing a different way of developing software (e.g., *Waterfall Model*, *Iterative Model*, *Spiral Model*, *V-shaped Model*, and *Agile Model*). A central task described by all SDLC models is the identification of the purpose for which the software is intended [96]. More concretely, we need to identify relevant stakeholders and understand their needs to build software that meets their requirements. RE aims at discovering the purpose of software and is, therefore, part of all mentioned SDLC models.

Requirements Engineering Activities The RE process consists of five core activities: elicitation, analysis, documentation, validation, and management of requirements [32, 31, 97]. In the following, we outline these activities and explain how conditionals are embedded in the RE process.

Elicitation This activity involves the search, collection, and consolidation of requirements from available sources. Potential sources for requirements are stakeholders who will use the software under development, as well as existing documents and prototypes. The result of the elicitation step is a list of requirements reflecting the perception of the desired functionalities gained by the requirements engineer. The requirements may therefore contain defects if the understanding of the requirements engineer is not in line with the understanding of the stakeholders.

Analysis During this step, the elicited requirements are reviewed jointly with the stakeholders to ensure that the collected requirements correspond to the actual needs of the stakeholders. Furthermore, conflicts between the requirements are identified and resolved. The deliverable of this step is thus a set of consolidated, non-contradictory requirements.

Documentation In this step, the set of consistent requirements is documented in the form of RE artifacts. For this purpose, different types of RE artifacts like use cases, user stories, goal models, or traditional software specifications can be used.

Validation This step checks whether the created RE artifacts comply with certain quality standards (e.g., ISO 29148 [98]) and verifies whether the three activities described above have been executed properly. The validation activity constitutes therefore a cross-cutting operation.

Management Requirements management deals with maintaining RE artifacts across their life cycle, organizing RE activities, and observing the system context to discover changes that require assistance from RE. Similar to the validation step, requirements management represents a cross-cutting activity.

Our studies reveal that conditionals are a widely used linguistic pattern to express intended system behavior (see Chapter 4). Accordingly, conditionals are inherent to requirements elicitation, since conditionals are intuitive means for stakeholders to describe

expected inputs and desired system behavior (e.g., *"If input X, the system shall . . . "*). Ultimately, conditionals are also included in the documented RE artifacts. In fact, we found that conditionals are prevalent in both traditional RE artifacts [4] as well as agile requirements such as acceptance criteria [1]. This thesis focuses on the automated conditional extraction from RE artifacts and thus builds on the output of the documentation activity.

Conditional Extraction in the Software Development Process As described in Section 1.2, automated conditional extraction from requirements contributes to the automation of two use cases, namely *"acceptance test creation"* (Use Case 1) and *"dependency detection between requirements"* (Use Case 2). These use cases fit into different parts of the software development process. Use Case 1 deals with acceptance testing which aims to determine whether a system fulfills end-user requirements. More specifically, acceptance testing can be understood as *"formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers, or other authorized entity to determine whether to accept the system"* [99]. We extract conditionals from documented requirements and create corresponding acceptance tests automatically. Hence, we support test designers, requirements engineers, and domain experts, who collaborate on creating acceptance tests [100, 101]. Acceptance tests are usually written in NL and are mostly executed manually [102, 103]. Each acceptance test contains a finite set of test cases that specify certain test inputs (input parameters) and expected results (output parameters) [104]. Each input and output parameter is defined by a variable and a condition that the parameter can take [105]. For example, the parameter *the system detects an error* can be decomposed into *"**variable**: the system"* and *"**condition**: detects an error"*. All test cases that constitute a single acceptance test are summarized in a test case specification. Each row represents a test case. The variables of the input and output parameters are listed in the columns. The conditions of the parameters that shall be inspected as part of a certain test case are contained in the respective cells. Contrary to Use Case 1, which bridges the gap between RE and acceptance testing, Use Case 2 is part of the RE process. That is, automated conditional extraction helps requirements engineers to detect dependencies between the requirements (e.g., conflicts and redundancies) and thus supports both the analysis and validation activity in the RE process.

2.3 Relevant Natural Language Processing Methods

Since NL is still the dominant notation style for requirements, our extraction approach must be able to understand natural language. Therefore, we use several NLP methods to implement our approach in the course of this thesis. NLP is a research discipline that bridges linguistics and ML and deals with the automatic processing of unstructured, natural language data [106]. This section provides knowledge about relevant NLP methods to an extent that enables readers to follow our approaches described in Part II.

2.3.1 Dependency Parsing

There are two main approaches suitable for describing the structure of a sentence [107]. The sentence is either separated into individual constituents (*Constituency Parsing*) or

its lexical units (words) are connected to each other forming the whole structure of the sentence (*Dependency Parsing*). These word-to-word links are defined as dependencies that specify the relationship between the words. Compared to *Constituency Parsing*, *Dependency Parsing* has a number of advantages and therefore constitutes the foundation for the modern, computer-based interpretation of language. It is robust against flexible word order languages [108] and its dependencies serve as a good basis for the subsequent semantic analysis of the sentence [107]. In Section 7.1, we utilize the potential of *Dependency Parsing* and analyze the dependencies between the words to detect *antecedents* and *consequents* in an NL sentence. The idea of *Dependency Parsing* was first introduced by Tesnière [109]: If two words are related, one word is defined as *head* and the other as *dependent*, which are connected by a directed link representing their syntactic relationship (*Dependency Relation*). This relationship can be expressed as a dependency triple $d(v_i, v_j)$ consisting of three parts [110]. v_i and v_j are two words connected by the relation d, where v_i is the *head* and v_j is the *dependent*. The sum of all dependency triples can be represented as a directed rooted tree. For each word, a node is created and each relation is represented by an arc in the tree, as in the dependency parse tree shown in Figure 2.3.

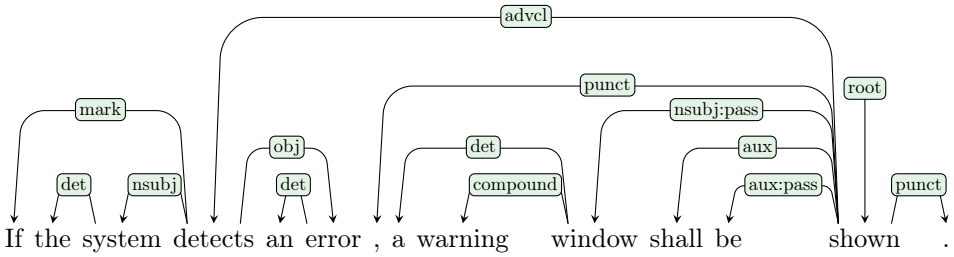


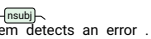
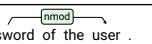
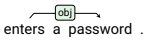
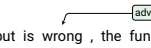
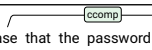
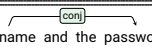
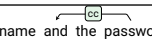
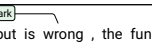
Figure 2.3: Exemplary Dependency Parse Tree.

Here the token *"shown"* represents both the root of the dependency tree and the predicate of the independent clause *"a warning window shall be shown"*. The relation $advcl(shown, detects)$ indicates that *"shown"* is modified by the dependent clause *"If the system detects an error"*. More specifically, the adverbial clause specifies *when* the action of the independent clause shall take place. In this case, *"shown"* is the *head* and *"detects"* is the *dependent*. Modern dependency parsers are capable of identifying a multitude of different relationship types between words. A rich overview of these relationships can be found in the *Universal Dependencies* project [111]. In Table 2.1, we give an insight into dependency types that are mainly relevant for our conditional extraction approach.

Transition-Based vs. Graph-Based Dependency Parsing In general, dependency parsers are implemented by either transition-based or graph-based approaches [112]. Transition-based *Dependency Parsing* involves four main components: (1) a stack on which the parse is built, (2) a buffer consisting of all remaining tokens to be parsed, (3) a parser that walks through a sentence from left to right and shifts tokens from the buffer into the stack, and (4) an oracle that examines the two top elements of the stack and predicts one of the following transitions [107, 113]:

- **Transition 1:** Create a dependency relation where the word at the top of the stack is the head and the second word is the dependent. Further, remove the second word from the stack.

Table 2.1: Overview of Universal Dependency Relations Relevant for Automated Conditional Extraction, Taken Verbatim From `https://universaldependencies.org/`.

Nominals	nsubj	A nominal subject (nsubj) is a nominal which is the syntactic subject and the proto-agent of a clause.	
	nmod	The nmod relation is used for nominal dependents of another noun or noun phrase and functionally corresponds to an attribute, or genitive complement.	
	obj	The object of a verb is the second most core argument of a verb after the subject. Typically, it is the noun phrase that denotes the entity acted upon or which undergoes a change of state or motion (the proto-patient).	
Clauses	advcl	An adverbial clause modifier is a clause which modifies a verb or other predicate as a modifier not as a core complement. This includes temporal clauses, consequences, conditional clauses, purpose clauses, etc.	
	ccomp	A clausal complement of a verb or adjective is a dependent clause which is a core argument. That is, it functions like an object of the verb, or adjective.	
Coordination & Modifiers	conj	A conjunct is the relation between two elements connected by a coordinating conjunction, such as and, or, etc.	
	cc	A cc is the relation between a conjunct and a preceding coordinating conjunction.	
	mark	A marker is the word marking a clause as subordinate to another clause.	
	amod	An adjectival modifier of a noun (or pronoun) is any adjectival phrase that serves to modify the noun (or pronoun).	

- **Transition 2:** Create a dependency relation where the second word of the stack is the head and the top word is the dependent. Further, remove the top word from the stack.
- **Transition 3:** Shift the current word in the buffer into the stack.

The parser terminates when the buffer is empty and the stack contains only a single word (arc-standard transition system). To enable the oracle to predict which of the three transitions should be executed, it is trained on gold standard dependency treebanks (e.g., *Penn Treebank* [114]). A dependency treebank is a corpus of NL sentences annotated with the corresponding dependency trees. The dependency treebanks are either a result of manual annotation or are created by converting existing constituent-based treebanks into dependency trees [112].

Graph-based *Dependency Parsing* does not process a sentence word by word to build a corresponding parse tree. Rather, it considers all possible trees for a given sentence and aims to identify the tree that maximizes some score. The search space of all potential dependency trees is defined as a fully-connected, weighted, directed graph of a given sentence [112]. Its vertices are the words of the sentence and the directed edges represent all possible dependency relations. Graph-based dependency parsers use methods [115, 116] from graph theory to find the highest scoring tree in the search space (i.e., the maximum spanning tree). In this context, graph-based methods rely - similarly to transition-based methods - on an oracle that parameterizes the search space by weighting the edges of the fully-connected graph. To predict the weights of the edges resp. the probability of a certain dependency relation, the oracle is developed using traditional supervised machine learning techniques (e.g., *Decision Trees*, *Support-Vector Machines*, *Neural Networks*) and trained on annotated dependency treebanks.

Studies [117, 118] show that both approaches have their individual advantages and disadvantages. For example, transition-based methods allow highly efficient parsing due

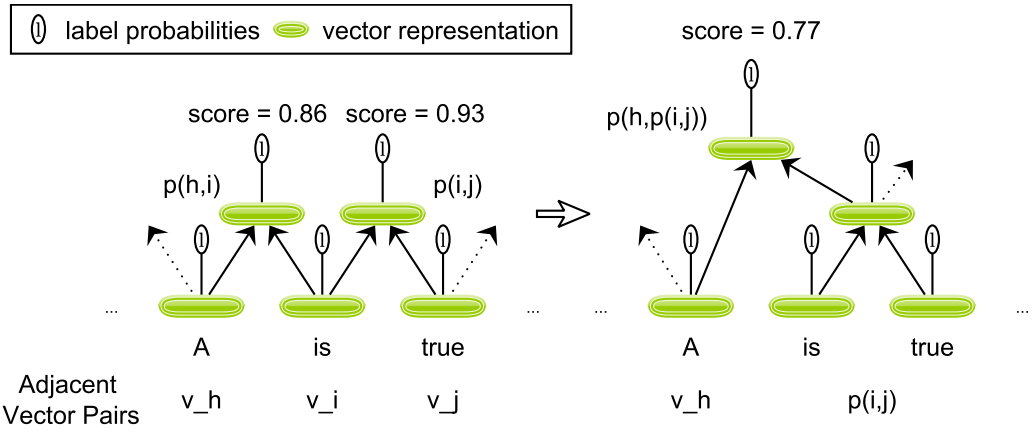


Figure 2.4: Bottom-Up Approach of an RNTN for the Prediction of a Binary Parse Tree, Based on [128].

to linear time complexity, but struggle when the *heads* are very far from the *dependents*. Graph-based methods are more accurate in parsing long sentences, but show slower inference time. Nevertheless, both approaches have proven to be suitable for building powerful dependency parsers [119, 120, 121]. For example, the *MaltParser* [122] and the *spaCy* dependency parser [123] are based on transition-based methods. Graph-based models are used by the *MSTParser* [124], *TurboParser* [125], and *Deep Biaffine* [126]. In this thesis, we use the *spaCy* NLP library v3.0 [123] to build dependency trees since it achieves state of the art results when evaluated on the *Penn Treebank* [127].

2.3.2 Recursive Neural Tensor Networks

An RNTN is a special type of *Neural Network* and has been invented by Socher et al. [128]. It is based on the idea that NL can be understood as a recursive structure. For example, the syntax of a sentence is recursively structured, with noun phrases containing relative phrases, which in turn contain further noun phrases, and so on. An RNTN is capable of recovering this recursive structure and helps to better understand the composition of a sentence. We argue that a conditional also represents a recursive structure as it consists of *antecedents* and *consequents*, which in case of conjunctions and disjunctions consists of further *antecedents* and *consequents*, and so on [2]. Building on this idea, we train an RNTN for conditional extraction (see Section 7.2). In this section, we explain its characteristics with respect to forward and backward propagation.

Let us define a sentence as a sequence of words $(w_i, ..., w_n)$, each represented by a corresponding d-dimensional vector v_i. The concept of an RNTN is to identify related vectors and merge them into pre-defined segments. We train the RNTN, for example, to demarcate vectors that describe *antecedents* or *consequents* from vectors that are not relevant for automatic test case derivation. To illustrate the composition of a sentence, an RNTN builds up a binary tree of segments in a bottom-up fashion (see Figure 2.4).

Forward Pass At the beginning of each recursion, the RNTN determines a set of all adjacent vector pairs $A = \{[v_i, v_j] | v_i \text{ and } v_j \text{ are adjacent}\}$. For each adjacent pair, the

RNTN computes: a parent representation $p_{(i,j)}$ and its label probabilities l. The parent representation is calculated by merging v_i and v_j according to the following equation:

$$p_{(i,j)} = f\left(\underbrace{\begin{bmatrix} v_i \\ v_j \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} v_i \\ v_j \end{bmatrix}}_{\text{Enhancement in RNTN}} + \underbrace{W \begin{bmatrix} v_i \\ v_j \end{bmatrix}}_{\text{Vanilla RNN}} \right) \tag{2.4}$$

An RNTN stores its trainable parameters in a weight matrix $W \in \mathbb{R}^{d \times 2d}$ and a tensor $V \in \mathbb{R}^{2d \times 2d \times d}$. In vanilla *Recursive Neural Networks* (RNN) [128], p_i is computed only using W. The concatenated v_i and v_j are simply multiplied with W and then given into an activation function f. This has the drawback that the input vectors only implicitly interact through the nonlinearity activation function. Consequently, the meanings of the words do not actually relate to each other and the influence of a child node on a parent node can not be adequately captured. However, this is especially important for the extraction of conditionals. For example, the model should learn that *"if"* usually indicates an *antecedent* segment, while *"then"* denotes a *consequent* segment. To overcome this issue, Socher et al. [129] introduced a three-dimensional tensor V into the model. Firstly, V allows more interactions between the vectors through the more direct multiplicative relation. Secondly, V adds another fixed number of parameters to the model, which allows the RNTN to gather even more information about the composition of a sentence. Each slice of the tensor is capturing a specific type of composition making the RNTN capable of understanding the structure of a sentence [129]. Each $p_{(i,j)}$ is then given to a softmax classifier to compute its label probabilities l. In other words, the RNTN calculates the probability for $p_{(i,j)}$ representing a certain segment. For classification into three segments (e.g., *antecedent*, *consequent*, not relevant), we compute the softmax score as follows:

$$l = \text{softmax}(Cp_{(i,j)}) \tag{2.5}$$

$C \in \mathbb{R}^{3 \times d}$ is the classification matrix. After computing the probabilities for all adjacent pairs, the RNTN selects the pair which received the highest softmax score and updates A by removing v_i and v_j and adding $p_{(i,j)}$ (see Figure 2.4). This process is repeated until all adjacent pairs are merged and only one vector is left in A. This vector represents the whole sentence and corresponds to the root of the constructed tree-structure.

Backward Pass We want to maximize the probability that the RNTN correctly predicts the *antecedents* and *consequents* in a sentence. For this purpose, we minimize the cross-entropy error between the predicted and target labels at each node. The backpropagation of an RNTN is slightly more complex than in conventional neural models because the errors have to be routed through the tree structure (starting from the root). *Backgropagation through structure* [130] is characterized by the following three properties: Firstly, the full derivative for V and W is the sum of the derivatives at each of the nodes. Secondly, the derivatives are split at each node and are sent down both branches of the tree to the next level below. Thirdly, for each node, the error message is the sum of the error propagated from the parent and the error from the node itself. For a detailed description of *backpropagation through structure*, please refer to the paper by Goller and Küchler [130].

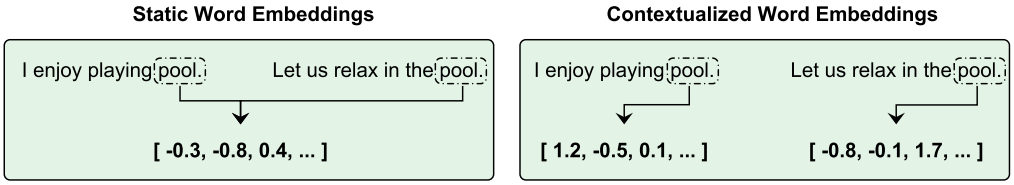


Figure 2.5: Difference Between Static (Left) and Contextualized Word Embeddings (Right).

2.3.3 Transfer Learning

Driven by the recent advances of *Deep Learning* (DL), more and more researchers are using DL models for NLP tasks. In this context, the *Bidirectional Encoder Representations from Transformers* (BERT) language model [131] is prominent and has already been used for *Question Answering* [132], *Named Entity Recognition* [133], and *Text Summarization* [134]. BERT is pre-trained on large corpora and can therefore easily be fine-tuned for any downstream task without the need for much training data. Specifically, BERT acquired a deep language understanding through pre-training on the *BookCorpus* (800M words) and a version of the *English Wikipedia* (2,500M words), making it easier for BERT to learn the requirements of a downstream language understanding task [112, 131]. The concept of combining pre-training and fine-tuning is also known as *Transfer Learning* (TL): *"the method of acquiring knowledge from one task or domain, and then applying it (transferring it) to solve a new task"* [112]. TL models are therefore particularly valuable for use cases for which only little training data is available, or where the effort required to create new training data is unreasonably high.

In recent years, TL models became increasingly popular in the BERT community and demonstrated strong performance, e.g., in the classification of requirements [38], and mining feature requests and bug reports from tweets and app store reviews [12]. In this thesis, we investigate the performance of TL models in the detection (see Chapter 6) and fine-grained extraction (see Section 7.3) of conditionals from NL requirements. This section covers the foundations of modern TL models by introducing contextual word embeddings, the transformer architecture, and presenting examples of TL models such as BERT, RoBERTa [135], and DistilBERT [136] that we employ in the course of this work.

Input Representation To process NL with computational models, it must first be transposed into a vector representation [137]. Learning suitable representations of NL data has been an active area of research for decades [138, 139, 140]. The idea of using a vector space to represent the connotation of words originates from Osgood et al. [141]. A mapping of a word to its corresponding vector is also called *word embedding* (i.e., a word is encoded in the vector space). The goal of word embeddings is to reflect the meaning of a word by its distribution in language use. Specifically, *"words that occur in similar contexts tend to have similar meanings"* [112] and should therefore be mapped to similar locations in vector space. Hence, embeddings aim to characterize words by the words that occur with them.

Based on the idea that the similar distribution of words is associated with similar meaning (also known as *distributional hypothesis* [142]), a number of approaches for the derivation of word embeddings have been developed. For example, the *Term*

Frequency–Inverse Document Frequency (TF-IDF) approach uses simple co-occurrence statistics and builds word embeddings by counting nearby words. The dimension of the vectors corresponds therefore to the size of the used vocabulary. As a consequence, the word embeddings tend to be very long and contain mostly zeros, since many words never appear in the context of others [112]. This type of word embedding is often described as *sparse* vectors. To address the issue that the size of embeddings increases with growing vocabulary, multiple approaches (e.g., *Word2Vec* [139], *Global Vectors for Word Representation* (GloVe) [140], *FastText* [143]) for creating *dense* word embeddings have been developed. Instead of containing a series of zeros, *dense* vectors have a fixed dimension and contain real-valued numbers that can be positive or negative. *Word2Vec* uses a classifier to predict the probability of two words being neighbors and utilizes the weights of the trained classier as the word embeddings. The resulting embeddings are static, meaning that the model creates one fixed vector for each word in the vocabulary [112]. Static embeddings suffer from an obvious weakness: a word is always represented by the same vector, regardless of the context in which the word occurs (see Figure 2.5). This limits their ability to cover the meaning of words since words naturally have different meanings depending on their context (e.g., *"I enjoy playing* ***pool****"* vs. *"Let us relax in the* ***pool****"*). Approaches like BERT, *Embeddings from Language Model* (ELMo) and *Generative Pre-trained Transformer 2* (GPT-2) [144] solve this shortcoming and explicitly take the context of a word into account when creating embeddings. In essence, modern language models predict the probability that a certain sequence of words occurs and utilize the calculated hidden states as word embeddings. Hence, the contextualized embedding of a word depends on all other words in a sentence [112]. This allows to generate different word embeddings for the word *"pool"* in the example above (see Figure 2.5).

In the following, we focus on BERT and its successor models RoBERTa [135] and DistilBERT [136], as we use them for the fine-grained extraction of conditionals. To enable BERT to create contextualized word embeddings, the input sequence must first be converted into a suitable format. For this purpose, the sequence is split into individual components using the *WordPiece* tokenizer (see Figure 2.6). *WordPiece* is a subword-based tokenization algorithm and aims at covering an infinite vocabulary with a finite set of known words [145, 146]. Specifically, *WordPiece* addresses the drawbacks of word-based tokenization approaches that struggle to process OOV words. To this end, frequently used words are not split into subwords, while rare words are decomposed into lexical units that are known to the model. Thus, even OOV words can be processed by the model, since their respective subwords may contain enough semantic information enabling the model to infer the semantic meaning of the OOV word. For example, let us apply the *WordPiece* tokenizer to the name of the architecture on which BERT is based: *"Transformer"*. The word is considered a rare word and split into *"Transform"* and *"##er"*, where *"##"* indicates that the token belongs to the previous one. Although the word is unknown to the model, the model can still understand its semantics as it contains a frequently used word: the verb *"transform"*. As described in Section 7.3, we leverage the ability of the *WordPiece* tokenizer to process OOV words in order to increase the robustness of our conditional extraction approach.

When using the *WordPiece* tokenizer it must be considered that BERT requires input sequences with a fixed length (maximum 512 tokens) [131]. Therefore, for sentences that are shorter than this fixed length, *Padding* (PAD) tokens need to be inserted to adjust

all sentences to the same length. Other tokens, such as the *Classification* (CLS) token and the *Separator* (SEP) token, are also inserted in order to provide further information of the sentence to the model. CLS is the first token in the sequence and represents the whole sentence (i.e., it is the pooled output of all tokens of a sentence). SEP marks the end of the sentence. After adding these synthetic tokens to the tokenized sequence, each token is fed into BERT to generate a corresponding embedding (see Figure 2.6).

Transformer Architecture BERT is a transformer-based architecture. In its vanilla form, a transformer architecture consists of two parts: an *encoder* that converts an input (e.g., German text) into a vector representation, and a *decoder* that transforms the created vector into an output (e.g, English text). This separation reflects that the transformer architecture was originally designed for machine translation [147]. BERT is an *encoder-only* model and focuses on the derivation of contextual embeddings (see Figure 2.6). Specifically, it maps an input sequence $x_1 \dots x_n$ to a contextualized encoded sequence $y_1 \dots y_n$:

$$f_{\text{BERT}} : x_1 \dots x_n \mapsto y_1 \dots y_n \tag{2.6}$$

BERT consists of a stack of multiple encoders and is available in two versions: the $\textbf{BERT}_{\textbf{Base}}$ model has 12 encoder layers stacked on top of each other, whereas the $\textbf{BERT}_{\textbf{Large}}$ model includes 24 encoder layers. Each encoder consists of (1) a multi-head self-attention mechanism, and (2) a simple, position-wise fully connected feed-forward network [147]. The distinctive feature of BERT compared to other language models is its *bidirectionality*. Specifically, it considers both the left and right context of a word when creating a corresponding embedding [131]. Previous transformer models like GPT-2 process an input sequence only in a left-to-right fashion. Hence, the hidden states are computed independently of the others as the model only considers tokens seen earlier in the context (i.e., the embeddings only contain information of the *right* context). This issue is especially problematic when utilizing embeddings to solve complex NLP problems like the fine-grained extraction of conditionals. Sophisticated linguistic labels can only be assigned correctly if both the *left* and *right* context of the tokens are considered. For example, let us apply the $\textbf{BERT}_{\textbf{Base}}$ model to the following requirement: *"If the system detects an error, it shows a warning message."* To understand that the token *"it"* in the exemplary requirement is part of a *consequent* but not of an *antecedent*, we need to consider that *"it"* is not located in the if-clause (the *left* context) as well as that the token *"accordingly"* at the end of the clause refers to the presence of a *consequent* (the right context). The use of the self-attention mechanism enables BERT to understand such long-range dependencies between tokens and thus to overcome the vanishing gradient problem of sequential models. To this end, BERT applies the self-attention mechanism over the entire input sequence and contextualizes each token using information from the entire input (highlighted in Figure 2.6 for x_1 and x_{13}). The final embeddings have a size of 768 dimensions ($\textbf{BERT}_{\textbf{Base}}$) or 1024 ($\textbf{BERT}_{\textbf{Large}}$). The dense layer within the encoder is used to further enrich the output of the attention mechanism and to pass it to the next encoder.

Related BERT Models: RoBERTa and DistilBERT After the release of BERT, scientists conducted a series of replication studies to identify means to further increase the perfor-

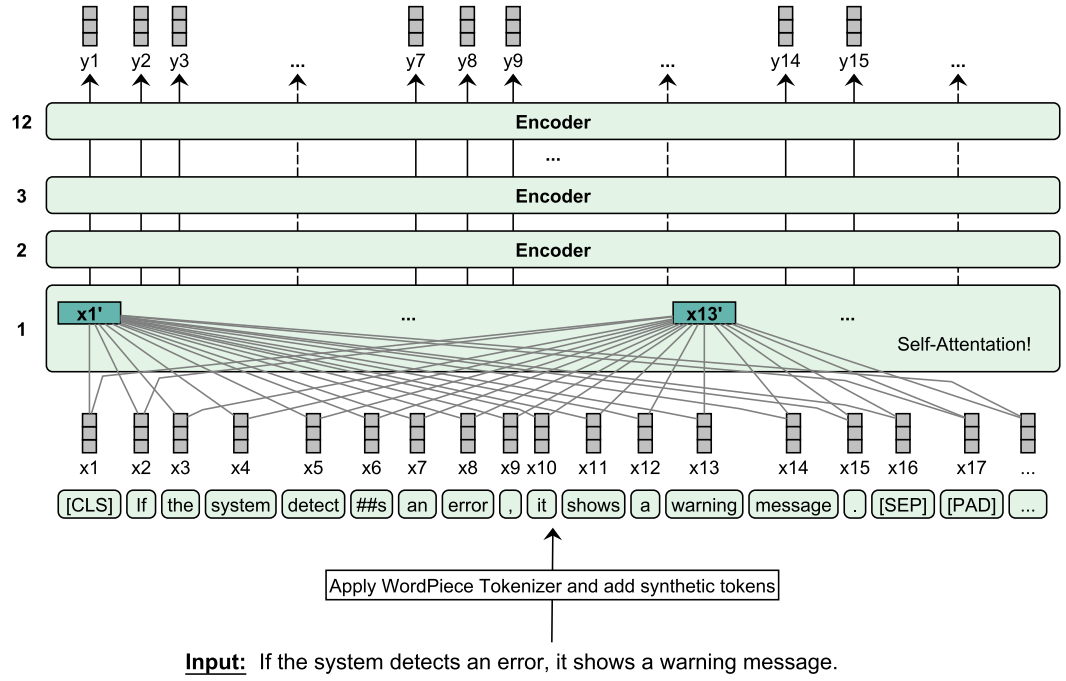


Figure 2.6: Overview of BERT Architecture.

mance of BERT. Liu et al. [135] found that longer pre-training with a higher batch size on an even larger data set can further improve the performance of BERT. Specifically, they train BERT on five English-language corpora of varying sizes and domains, totaling over 160 GB of uncompressed text. In addition, they discovered that the next sentence prediction objective during pre-training does not lead to a performance gain and therefore suggest to train BERT only by masked language modeling. Based on these adjustments, they introduce RoBERTa as a tuned version of BERT, which shows better prediction performance on various benchmarks but negatively affects training and inference time.

The pre-training of modern language models is computationally expensive. For this reason, Sanh et al. [136] investigated how to reduce the size of the BERT model while retaining its language understanding capabilities. They propose DistilBERT which represents the distilled version of BERT that allows for faster training. It retains 97 % of BERT performance and yields comparable performance on downstream tasks.

2.4 Cause-Effect-Graphing

Cause-Effect-Graphing aids in selecting the minimal number of required test cases by illustrating the desired system behavior in the form of input and output combinations [148, 149]. It belongs to the group of black-box testing methods since it only considers the external behavior of a system. A *Cause-Effect-Graph* (CEG) models inputs as *causes* and outputs as *effects*. A *cause* can be interpreted as a condition or a class of conditions that lead to a certain action of the system. *Effects* represent any output conditions produced by the system. A CEG can be considered as a combinatorial logic network, which

describes the interaction of *causes* and *effects* by Boolean logic [64]. Both *causes* and *effects* are represented as nodes and associated by four different relationships: conjunction $\wedge$, disjunction $\vee$, negation $\neg$ and identity $\Leftrightarrow$. The nodes in a CEG can either take the value 1 (*"present"* state) or 0 (*"absent"* state). Accordingly, the four relationships between *causes* (c_n) and *effects* (e_n) are defined as follows [148]:

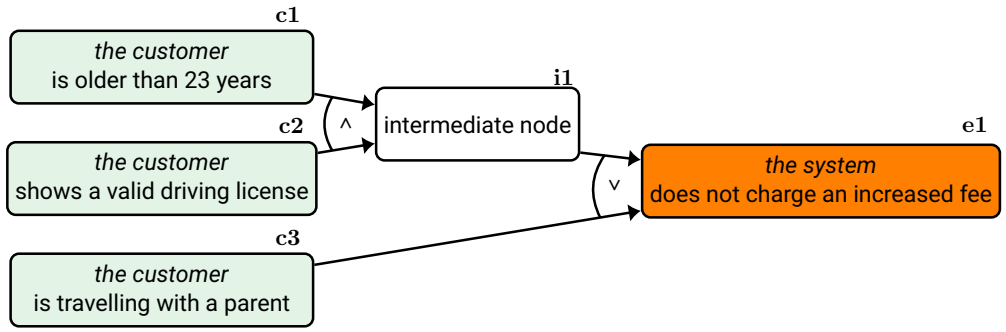
- $c_1 \Leftrightarrow e_1$: The identity relation states that if c_1 is 1, e_1 is also 1; else e_1 is 0. Hence, c_1 is both a *sufficient* and a *necessary* condition for the occurrence of e_1.
- $\neg\ c_1 \Leftrightarrow e_1$: The negation relation states that if c_1 is 1, e_1 is 0; else e_1 is 1.
- $c_1 \vee c_2 \Leftrightarrow e_1$: The disjunction relation states if c_1 or c_2 is 1, e_1 is 1; else e_1 is 0.
- $c_1 \wedge c_2 \Leftrightarrow e_1$: The conjunction relation states that if both c_1 and c_2 are 1, e_1 is 1; else e_1 is 0.

To demonstrate the core idea behind *Cause-Effect-Graphing*, we use the following requirement as a running example: *"If the customer is traveling with a parent, or the customer is older than 23 years and the customer shows a valid driving license, the system does not charge an increased fee."* We model the requirement as $G = (E, C)$ shown in Figure 2.7a with effect set E and cause set C. In the example, $|C| = 3$ including $c1$, $c2$, and $c3$ while $|E| = 1$ with $e1$. Please note that the intermediate node is required to model the combination of conjunction and disjunction.

Basic Path Sensitization Technique To derive test cases from our constructed CEG, the *Basic Path Sensitization Technique* (BPST) is applied [64]. BPST derives test cases from a CEG in four steps. Specifically, it converts a CEG into a decision table, where each column of the decision table corresponds to a test case:

- **Step 1:** Select any *effect* in the CEG.
- **Step 2:** Scan the CEG for combinations of *causes* that do or do not lead to the occurrence of the selected *effect*.
- **Step 3:** Create one column in the decision table for each of the identified combinations of *causes* as well as the resulting states of the other *effects*.
- **Step 4:** Check whether entries in the decision table occur twice and remove them if necessary.

The determination of *cause* combinations mentioned in the second step is performed by traversing the CEG back from the *effects* to the *causes* according to specific decision rules, which were originally introduced by Myers [148] and then fine-tuned by Liggesmeyer [150] and Nursimulu and Probert [64]. We describe these rules in detail below, as *Cause-Effect-Graphing* is an integral part of our approach to automatic acceptance test generation (see Chapter 10). Please note that a CEG, unlike its name implies, does not model a causal relation according to the definition presented in Section 2.1.1. Its purpose is rather to express the dependency between different inputs and outputs, which lends itself to the modeling of conditionals. The reason for naming the nodes as *causes* and *effects* was presumably to distinguish more clearly between input and output variables.



(a) *Cause-Effect-Graph. Antecedents* Are Represented as Cause Nodes (Highlighted in Green). *Consequents* Are Illustrated by Effect Nodes (Highlighted in Orange).

		Test Cases				
		TC 1	TC 2	TC 3	TC 4	TC 5
Input	c1	1	1	0	1	0
	c2	1	0	1	0	1
	c3	0	1	1	0	0
	i1	1	0	0	0	0
Output	e1	1	1	1	0	0

(b) Minimal Test Suite Automatically Created by BPST

Figure 2.7: Application of *Cause-Effect-Graphing* for the Automatic Derivation of Test Cases From the Requirement: *"If the customer is travelling with a parent, or the customer is older than 23 years and the customer shows a valid driving license, the system does not charge an increased fee."*

- **Rule 1:** In case of disjunctions with the result 1 (i.e., the *effect* occurs), we only build input combinations, where one input has the value 1 and all other inputs have the value 0.
- **Rule 2:** In case of disjunctions with the result 0 (i.e., the *effect* does not occur), we set all inputs to 0.
- **Rule 3:** In case of conjunctions with the result 0 (i.e., the *effect* does not occur), we only form input combinations where one input has the value 0 and all other inputs have the value 1.
- **Rule 4:** In case of conjunctions with the result 1, we set all inputs to 1.

These rules achieve the maximum probability of finding failures while avoiding the complexity of generating 2^n test cases, where n is the number of causes [150]. Thus, *Cause-Effect-Graphing* enables balancing between sufficient test coverage and the lowest possible number of test cases. This can be illustrated by applying the described rules to our constructed CEG shown in Figure 2.7a. We select e_1 as our entry point (see step 1). First, we search for all *cause* combinations where e_1 occurs. e_1 is caused by a disjunction of c_3 and i_1. According to rule 1, we insert into the decision table only the input combinations where one input has the value 1 and all other inputs have the value 0. Hence, we have to

consider the two cases $i_1 = 1$ and $c_3 = 0$ as well as $i_1 = 0$ and $c_3 = 1$. We do not need to handle the case $i_1 = 1$ and $c_3 = 1$ because of rule 1. i_1 is the result of a conjunction of c_1 and c_2. According to rule 4, we set c_1 and c_2 to 1 and thus include our first test case (TC 1) in the decision table. Since the CEG contains only one *effect* (e_1) we do not have to respect the resulting states of other *effects* in step 3. For the case $i_1 = 0$ and $c_3 = 1$ we have to apply rule 3, namely we only form input combinations where one input has the value 0 and all other inputs have the value 1. Therefore we have to consider the cases $c_1 = 1$ and $c_2 = 0$ as well as $c_1 = 0$ and $c_2 = 1$. This results in TC 2 and TC 3 in our decision table.

After the creation of the positive test cases, we also have to determine the negative test cases (i.e., the *cause* combinations under which e_1 does not occur). According to rule 2 we set i_1 and e_1 to 0. Since i_1 is caused by a conjunction of c_1 and c_2, we have to follow rule 3 for the case $i_1 = 0$. This results in the two cases $c_1 = 1$ and $c_2 = 0$ as well as $c_1 = 0$ and $c_2 = 1$ and consequently in TC 4 and TC 5 in our decision table. Since none of the entries in the decision table occur twice, we do not need to remove any entries and have completed the entire BPST process. Figure 2.7b contains our created test suite for G. To check the functionality presented in the CEG comprehensively, only five test cases are needed instead of the maximum number of 2^3 test cases.

Chapter 3

State of the Art

This chapter summarizes existing work in the research area of extracting conditionals from NL requirements artifacts for automatic acceptance test creation. More specifically, it reviews studies on the notion and logical interpretation of conditionals, presents existing approaches for the automated extraction of conditionals from NL, and describes existing methods for automated test case derivation from requirements. Each section outlines existing work, identifies open issues, and points to the chapters in this thesis that contribute to their resolution. The structure of this chapter reflects the organization of this thesis:

Section 3.1 outlines work on the prevalence of conditionals in the SE domain and conducted experiments related to conditional reasoning. The presented work provides evidence that conditionals are present in a variety of development artifacts (e.g., test cases and code). However, existing studies do not provide an overview of the extent, form, and complexity of conditionals in NL requirements, hindering the development of approaches capable of extracting conditionals from RE artifacts. We address this research gap in Chapter 4. Further, our literature review reveals a number of studies that show that people often deviate from the prescriptions of logic when interpreting conditionals. The contributions of our thesis presented in Chapter 5 support this evidence and additionally indicate that there is a statistically significant relation between the interpretation of conditionals and certain context factors of RE practitioners (e.g., experience in RE).

Section 3.2 outlines work on approaches for the automated extraction of conditionals from NL. We show that existing approaches are not capable of extracting conditionals in fine-grained form, rendering them unsuitable for our use case. Specifically, existing approaches only extract conditional pairs or detect conditionals on the phrase level while not considering the combinatorics between *antecedents* and *consequents*. We address this research gap in Chapter 6, Chapter 7, and Chapter 8 and present our tool-supported approach `CiRA` capable of detecting conditionals in NL requirements and extracting them in fine-grained form. A live demo of `CiRA` can be accessed at `www.cira.bth.se/demo/`.

Section 3.3 outlines work on approaches for the automated extraction of test cases from requirements. We show that most of the existing approaches allow the derivation of test cases from semi-formal or formal requirements, but are not suitable to process informal requirements. Some approaches address this research gap and focus on deriving test cases from informal requirements. Nevertheless, they have major drawbacks preventing their use in practice. For example, they do not ensure that only the minimal number of required test cases is created. In Chapter 10, we present an approach capable of deriving acceptance tests from conditional statements in NL requirements automatically. Our approach leverages `CiRA` to extract conditionals in fine-grained form and then maps the extracted conditionals into a CEG, from which we derive the minimal number of required test cases.

3.1 The Notion of Conditionals Statements

How do people formulate and understand conditionals? Linguists, mathematicians, and philosophers have been debating this question in their work for decades. There is still no consensus on how conditionals should be interpreted logically. Bryne and Johnson-Laird [151] therefore speak of sentences with *"if"* being a puzzle that has not yet been solved. In this section, we review existing work on the prevalence of conditionals (see Section 3.1.1) and logical interpretation of conditionals by humans (see Section 3.1.2).

3.1.1 Prevalence of Conditionals

Conditionals serve as an intuitive means of expressing actions and their consequences. Hence, conditionals occur frequently in everyday conversations and written texts. In this section, we report on work that deals with the occurrence and usage of conditionals in the SE domain. The contribution of Chapter 4 builds upon and expands this state of the art.

Conditionals in Requirements Stakeholders utilize conditional statements to formulate their expectations of a system. Specifically, they describe certain conditions that may occur and the corresponding system behavior that they expect. To precisely express conditionals in requirements, Mavin et al. [152] developed the *Easy Approach to Requirements Syntax* (EARS) patterns that represent a semi-formal notation style of requirements. In other words, they dictate a certain structure that must be respected by the authors of requirements while allowing them to fill the structure in any way. Mavin et al. [152] propose the following two patterns for the definition of conditionals in requirements:

- **Event-driven requirements:** WHEN <optional preconditions> <trigger> the <system name> shall <system response>.
- **Unwanted behaviors:** IF <optional preconditions> <trigger>, THEN the <system name> shall <system response>.

In the case of *event-driven requirements*, the system response is required *when and only when* the stated event is detected at the boundary of the system. The semantics of the *unwanted behavior* is defined similarly: the system is required to achieve the stated system response *if and only if* the preconditions and trigger are true. Hence, Mavin et al. [152] define conditionals in requirements as equivalences. This makes the requirements clear and testable because both positive and negative scenarios are defined. Studies [153, 154] demonstrate that the EARS patterns have been adopted by many organizations worldwide as they help authors to control the ambiguity, complexity, and vagueness of requirements.

Conditionals in Test Cases Conditional statements are often used to define manual test cases [155]. In this context, they serve as a means of specifying a condition to be checked by a tester and an action to be executed by the tester should the condition occur. Hauptmann et al. [156] show that conditional clauses can lead to ambiguities when executing test cases and therefore render the results of test runs nondeterministic. This poses a major problem for the entire test process as test results become neither interpretable nor comparable. This quality defect (also known as *branches in test flow*)

can be illustrated using the following conditional: *"If the selected customer is older than 18 years, set the discount option to false"*. According to Hauptmann [155], this test case description is problematic because it only instructs the tester on what to do if the selected customer is older than 18 years. The negative case - that the selected customer is younger than 18 years - is not explicitly defined (e.g., by an else-statement). Consequently, the tester is forced to decide independently how to interpret the negative case. In other words, the test procedure is not predefined.

As shown in Section 3.1.2, the negative case of a conditional can be interpreted in various ways, which in turn may cause testers to execute the same test case differently. Hence, test cases should be free of branching logic to ensure a deterministically executable test suite [156]. Rather, conditionals shall be split into individual test steps: (1) verify that the customer is older than 18 years and (2) set the discount option to false. Additionally, the negative case shall be checked in a separate test case: (1) verify that the customer is 18 years old or younger and (2) set the discount option to true.

Conditionals in Program Coding Conditionals do not only occur in NL, but also constitute a basic control structure in programming languages. Specifically, conditionals allow programmers to dictate the behavior of a computer when certain conditions are met. In this context, conditionals can be expressed in arbitrary complexity. In the simplest form, conditionals are specified using the `if...then...else` syntax. The cue phrase `if` introduces the conditional statement and specifies the condition that needs to be tested. The `if` clause contains any code that will be executed *if and only if* the condition is true. The `else` clause defines code that will be executed *if and only if* the condition is false. Hence, conditionals in code are defined as logical equivalences. The complexity of `if...then...else` constructs can be extended by additional operators like `else...if`, `switch`, `do...while` loops, and by nestings.

Conditionals are an elementary part of any software. Early studies have shown that conditionals are used extensively in program code. For example, Elshoff [157] found in a study of 120 *PLI/I* programs that conditionals comprise 17.8 % of all used statements. Similar results were obtained in the studies by Knuth [158] and Saal [159] when analyzing *FORTRAN* and *APL* programs. Conditionals also play a crucial role in the development of modern systems and are supported by state of the art programming languages such as *Python*, *JavaScript*, *Swift*, *Scala*, and *Go*. Recent studies [160, 161] demonstrate that `if` statements are deemed important by development teams since they enable the definition of rules according to which software should operate. Due to constantly evolving requirements, conditionals are often changed during development projects [162, 163] and belong to the most frequently modified control structures in code reviews [164].

Summary and Relation to Our Thesis:

Existing studies prove that conditionals are present in a variety of artifacts used during the software development cycle: e.g., requirements, test cases, and code. So far, it is known that conditionals have played an important role in the development of legacy systems and still do when implementing modern systems.

However, we know little about the extent, form, and complexity of conditionals in NL requirements artifacts. Certainly, studies show that EARS patterns are used

by some companies to formulate conditionals but it is not reasonable to prepare our conditional extraction approach for processing requirements in EARS notation since NL is still the dominant notation style in practice [61, 62]. Thus, we need to understand how conditionals are expressed by authors in NL requirements and implement our approach accordingly. In this thesis, we address this research gap and investigate the prevalence of conditionals in NL requirements artifacts (see Chapter 4). We highlight that conditionals matter in both traditional requirements documents as well as agile RE artifacts such as acceptance criteria. Further, we demonstrate that most conditionals in RE artifacts are *explicit* and occur in *marked* form. We also show that they may include up to three *antecedents* and two *consequents*. As a result, we expand the state of the art by a comprehensive insight into the notion of conditionals in NL requirements artifacts.

3.1.2 Logical Interpretation of Conditionals

The semantics of conditionals has been debated in research for decades and there is still no consensus on how conditionals should be interpreted. In this context, philosophers, linguists, and mathematicians have developed a series of theories about the logical interpretation of conditionals, leading to the emergence of a dedicated field of research: *conditional reasoning*. An overview of all published theories is beyond the scope of this thesis. However, we give an insight into the main ideas and conducted experiments related to conditional reasoning. The contribution of Chapter 5 builds upon and expands this state of the art.

Conditional Reasoning According to Thompson [165], conditional reasoning *"entails drawing inferences about situations in which the occurrence of one event is conditional or contingent upon the occurrence of another event"*. In a conditional reasoning tasks, subjects are usually asked to indicate the validity of four inferences derived from a given conditional. Let us assume the following conditional: *"If the car is out of gas (event p), then it stalls (event q)"*. The *Modus Ponens* (MP) inference entails concluding q, given p. In other words, *"the car is out of gas"*; therefore *"the car stalls"*. The *Modus Tollens* (MT) inference entails concluding $\neg p$, given $\neg q$ (i.e., *"the car has not stalled"*; therefore, *"the car did not run out of gas"*). The *Denying the Antecedent* (DA) inference entails concluding from $\neg p$ to $\neg q$. In other words, *"if the car does not run out of gas, it will not stall"*. The *Affirming the Consequent* (AC) inference entails concluding from q to p (i.e., *"the car has stalled"*; therefore, *"the car has run out of gas"*). In logic, both MP and MT are defined as valid forms of inference, while the DA and AC inferences are considered to be *logical fallacies* and are thus defined as invalid [166]. More specifically, mathematicians specify the *correct* semantics of sentences in the form *"if p then q"* as a conditional truth function. This means that the truth of *"if p then q"* can always be determined by inspecting the truth values of *"p"* and *"q"* [167]. The material conditional $p \rightarrow q$ is defined to be true as long as there is no case in which the *antecedent* is true and the *consequent* is false.

However, multiple studies [88, 89, 90] reveal that people frequently deviate from this as *ideally* defined interpretation and, from a logical perspective, draw invalid inferences. In other words, their interpretation does not conform to the prescriptions of logic.

Wason's Selection Task Already in 1966, Wason was able to show that people commit logical errors in the interpretation of conditionals [168]. He conducted an experiment and presented the subjects with four cards. Each card showed a colored, geometric figure. Specifically, a red triangle (RT), a blue triangle (BT), a blue circle (BC), and a red circle (RC). The subjects were told that each card has a triangle on one side and a circle on the other side. Each figure was colored either red or blue. The subjects were given the conditional: *"Every card which has a red triangle on one side has a blue circle on the other side"* and were asked to select those cards which need to be turned over in order to determine whether the conditional is true or false.

Interestingly, most of the subjects selected either the RT card or both the RT and BC cards. Only a few chose the correct answer: the RT and RC cards. Only if a card shows both the red triangle and the red circle, the conditional is false. The experiment has been replicated by other researchers [169] and all studies yield the same result: most subjects draw the AC inference from the conditional. They interpret the sentence as a bi-conditional, implying that *"if a card shows a blue circle, then it has a red triangle on the other side"*. Accordingly, they select the BC card beside the RT card to verify the conditional. In conclusion, the subjects interpreted the conditional as an equivalence (i.e., $p \Leftrightarrow q$) rather than as an implication (i.e., $p \Rightarrow q$). Studies [170, 171, 172] show that there is a variety of factors that can cause these *illogicalities* in the interpretation by people. We report on some of these factors in the following paragraphs.

Logical Interpretation Depends on Available Knowledge of the Conditional Relationship Digdon [173] and Pollard [174] reveal that conditional reasoning is influenced by available knowledge of the conditional relationship. They show that the logical interpretation of conditionals depends on the perceived necessity and sufficiency of the relationship. In the case of necessary relationships, the *consequent* event occurs only when the *antecedent* event occurs (e.g., *"if the computer is connected to the power supply, then it is ready to boot up"*). In the case of sufficient relationships, the *consequent* always occurs when the *antecedent* occurs (e.g., *"if the car runs out of gas, then it stalls"*). Several studies [90, 175, 176] found that people are more likely to draw the AC and DA inferences in necessary relationships as opposed to sufficient relationships. We explain this tendency below, building on the examples and the discussion by Thompson [165].

Necessary Relationship:

a) *If the computer is connected to the power supply, then it is ready to boot up.*
   AC: The computer is ready to boot up. Is it plugged in? (**yes**)
   DA: The computer is not plugged in. Is it ready to boot up? (**no**)

Sufficient Relationship:

b) *If the car runs out of gas, then it stalls.*
   AC: The car has stalled. Did it run out of gas? (**maybe**)
   DA: The car has not run out of gas. Did it stall? (**maybe**)

We know that a computer can only start up if it is connected to the power supply. Thus, we are aware that conditional a) denotes a necessary relationship. When a computer is ready to boot up, we intuitively conclude that it is also connected to the power supply and therefore draw the AC inference from conditional a). Similarly, we reject the claim that the computer can be started if it is not connected to the power supply (DA inference).

We know that a car can stall due to a variety of reasons (e.g., engine failure, flat tires). Consequently, we are aware that besides the *antecedent "the car runs out of gas"*, there are other, alternative *antecedents* that can provoke the *consequent "the car stalls"*. Hence, we know that conditional b) represents a sufficient relationship and do not commit the AC and DA fallacies (see answers above). Markovits [177] demonstrates that subjects who are aware of alternative *antecedents* are in fact much less likely to draw AC and DA inferences than people who do not know the alternative. Further, Byrne [178] and Rumain et al. [179] show that the likelihood of subjects committing logical errors can be reduced by explicitly naming alternative *antecedents*.

Logical Interpretation Depends on Mood of Subjects and Emotionality of Contents Studies indicate that the mood of subjects affects conditional reasoning. For example, Oaksford et al. [180] performed the Wason's Selection task with different groups and induced positive and negative moods in the groups. Subsequently, they compared the groups' selections with a mood-neutral control group. They found that the subjects in both positive and negative moods were less likely to draw the correct logic inferences compared to the control group. A similar observation has been made by Melton [181].

An experiment conducted by Blanchette & Richards [182] reveals that the emotionality of contents also influences conditional reasoning. The subjects were more likely to commit logical fallacies when reasoning about emotional content (e.g., *"If someone is in a tragic situation, then she cries"*) rather than when reasoning about neutral content (e.g., *"If someone is an actor, then he is an extrovert"*). This finding confirms the results of an early study by Lefford [183]. Evidently, the studies mentioned are subject to a series of threats to validity (e.g., inaccuracies during mood manipulation), which are also explicitly mentioned by the authors [184]. However, the results of the experiments provide a first indication that mood and emotional content may influence conditional reasoning in a certain way.

Logical Interpretation Depends on Preferences The scientific debate about conditional reasoning has long been based on the assumption that goals and preferences do not impact the process of drawing logical inferences from conditionals [185]. However, studies proved that this presumption is wrong and demonstrated that preferences do have an influence on conditional reasoning. This can be illustrated by the following examples.

c) *If Alice is invited to the party, she'll buy a new dress.*
MP: Alice is invited to the party. Will she buy a new dress? (**yes**)

From a logical viewpoint, Alice will certainly buy a new dress (MP inference). Evans et al. [186] presented subjects with conditional c) and asked them whether they would draw the MP inference. The subjects almost unanimously agreed and thus did not deviate from the prescriptions of logic (see answer above). In a follow-up experiment, conditional d) (see below) was added.

c) *If Alice is invited to the party, she'll buy a new dress.*
d) *If Alice buys a new dress, she can't pay the rent next week.*
MP: Alice is invited to the party. Will she buy a new dress? (**no**)

In this scenario, the participants denied that Alice intends to buy a new dress. The subjects assumed that Alice wants to pay her rent and does not want to take any action that would prevent her from doing so. Hence, the subjects did not draw the MP inference and deviated from the rules of logic because they expected Alice to act rationally and not decide against her preferences. The fact that preferences and goals matter in the interpretation of conditionals has been demonstrated by a number of further experiments [187, 188].

Summary and Relation to Our Thesis:

The wealth of published studies underlines that the semantics of conditionals in NL is widely discussed in the scientific community. At present, it is common sense that human reasoning and strict logical rules often do not coincide and that the logical interpretation of conditionals is influenced by a variety of factors.

In this thesis, we investigate how practitioners logically interpret conditionals embedded in requirements (see Chapter 5). Similar to the aforementioned studies, our results indicate that RE practitioners often deviate from the prescriptions of logic. In fact, nearly half of our respondents drew AC inferences when interpreting conditionals in requirements artifacts. We also found that RE practitioners disagree about the temporal occurrence of *antecedent* and *consequent* when different cue phrases (e.g., *"when, if, after"*) are used to express a conditional. Hence, a generic formalization of conditionals in RE artifacts will inevitably fail at least some practitioner's interpretation. This confirms the findings by Edgington [189] that a logical model is often not suitable to capture the interpretations of humans.

Our study also shows that there is a statistically significant relation between the interpretation of conditionals and certain context factors of RE practitioners. For example, the experience in RE as well as the way how a practitioner usually interacts with requirements (e.g., writing requirements vs. reading and implementing requirements) affects conditional reasoning. This thesis thus extends the literature with further factors that impact the logical interpretation of conditionals. Interestingly, we found that domain knowledge does not promote a consistent interpretation of conditionals in RE artifacts. Even in the case of close familiarity with a domain, some of our respondents interpreted conditionals in different ways.

3.2 Automated Extraction of Conditionals from Natural Language

Several approaches for automated conditional extraction have been developed. According to Yang et al. [190], these approaches can be grouped into three categories: knowledge-based, statistical *Machine Learning* (ML)-based, and *Deep Learning* (DL)-based approaches. The extraction of conditionals from NL consists of two steps. First, it must be detected whether an NL sentence contains a conditional. Second, the conditional (if present) has to be extracted. In this section, we give an overview of both approaches to conditional detection (step 1) as well as conditional extraction (step 2). To this end, we build on the systematic literature mapping studies conducted by Asghar et al. [191] and Yang et al. [190] who present a comprehensive insight into the state of the art in automated conditional extraction. As described in Section 1.2, we need to extract condi-

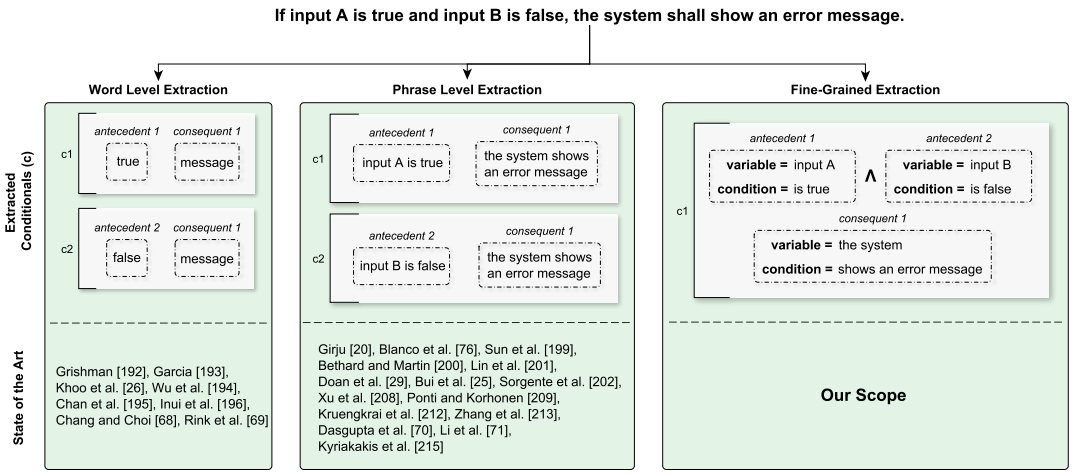


Figure 3.1: State of the Art in Automated Conditional Extraction.

tionals in fine-grained form to transfer them into a CEG. Hence, we describe not only the functionality of the existing methods for conditional extraction, but also the granularity (e.g., word level, phrase level) in which they extract the conditionals. Figure 3.1 illustrates the state of the art in automated conditional extraction from NL.

3.2.1 Knowledge-Based Approaches

Knowledge-based approaches detect and extract conditionals by applying linguistic patterns such as *"[antecedent] and because of this, [consequent]"*. Hence, their performance relies on hand-coded patterns, which require extensive manual work.

In early work, Grishman [192] proposes the *PROtotype TExt Understanding System* (PROTEUS) that can be used to identify conditionals in equipment failure messages. PROTEUS performs four steps: syntactic analysis, clause semantics, noun phrase semantics, and discourse analysis. Garcia [193] presents COATIS, a tool capable of identifying conditional statements in a single NL sentence written in French. He designs lexico-syntactic patterns based on 23 verbs that usually indicate conditionals like *"provoke"*, *"disturb"*, *"result"*, and *"lead to"*. Other early approaches are rooted in the medical domain, where relationships between symptoms and diseases are commonly expressed in natural language sentences utilizing conditionals: Khoo et al. [26] extract conditionals from a medical database using graphical patterns. The roles and attributes of a conditional are structured in a three-layer template, which constitutes the framework for manually elicited patterns. Wu et al. [194] show how to mine depressive symptoms from consultation records by using a rule-based system for conditional detection.

In the field of economics, conditional detection has been applied to improve the reasoning about market-related relationships. Early approaches include Chan et al. [195] utilizing a hierarchy of manually generated *semantic*, *sentence*, *consequence*, and *reason* templates. Other approaches as proposed by Inui et al. [196], which too extract conditionals from newspapers, base their conditional detection algorithm on the occurrence of certain cue phrases (e.g., *"if"*, *"when"*).

As indicated by Figure 3.1, all knowledge-based approaches extract conditionals only on word-to-word level. From sentences such as *"A wrong user input results in a restart of the system"*, they extract the *antecedents* and *consequents* only as single-word representations (**antecedent$_1$**: *"input"* and **consequent$_1$**: *"restart"*) resulting in the loss of essential information. For the creation of test cases, *antecedents* and *consequents* need to be described as a combination of variable (**antecedent_variable$_1$**: *"user input"*) and condition (**antecedent_condition$_1$**: *"wrong"*). Hence, we need to extract the entire embedded conditional statement (see Section 1.2).

3.2.2 Statistical Machine Learning-Based Approaches

Knowledge-based methods rely heavily on hand-crafted features and patterns, making them domain-dependent and reducing their generalizability. Additionally, the manual development of features and patterns is laborious and requires domain expertise. Therefore, research on conditional extraction shifted towards the usage of statistical ML techniques that can be trained to classify conditionals by using annotated data. Specifically, researchers employed third-party NLP tools (e.g., *Spacy* [123], *Stanford CoreNLP* [197]) to automatically generate a set of features for a given a data set, and then use ML algorithms (e.g., *Support Vector Machine*, *Logistic Regression*) to perform the classification task [190]. Hence, ML-based approaches require careful feature engineering as their performance depends on the manual selection of textual features.

Girju [20] proposes an approach using lexico-syntactic patterns within one sentence or two adjacent sentences. The patterns consist of two *Noun Phrases* (NP) connected with a causative verb in the following structure:

$$< NP_1, verb, NP_2 > \tag{3.1}$$

The patterns are built by traversing *WordNet* concepts for noun phrases that are connected by a *cause-to*-relationship, which is explicitly annotated in the *WordNet* corpus. Subsequently, from a large NL corpus, all verbs connecting these causally related noun phrases are extracted as causation verbs. Based on this information and further semantic features from *WordNet*, the lexico-syntactic patterns are created. Girju uses C4.5 decision tree learning [198] to build a system capable of detecting conditionals based on the developed patterns. Chang and Choi [68] expand on this concept by taking into account conceptual pair probability and cue phrase probability as additional indicators for the classification of a conditional statement. Specifically, they apply a *Naive Bayes* classifier to predict the probability of a conditional given a certain cue phrase (e.g., causative verb).

Blanco et al. [76] also focus on identifying explicit conditionals expressed by the pattern $< VerbPhrase, relator, Cause >$ where the relator is $\in \{because, since, after, as\}$. They apply a C4.5 decision tree binary classifier to determine sentences that match this pattern. Sun et al. [199] propose a different approach to extract conditionals from engine query logs. They use geometric features of events and then use the *Granger Causality Test* to re-rank them. Bethard and Martin [200] train different machine learning models using features derived from *WordNet* and the *Google* N-gram corpus. Their approaches achieve an F_1 score of 52.4 % in the extraction of conditionals and outperform other baseline systems. However, the obtained performance is still insufficient from a practical point of view and prevents the approaches from being used in the field.

More recent approaches like the one proposed by Rink et al. [69] use a *Support Vector Machines* classifier trained on contextual features. Lin et al. [201] train a classifier on the *Penn Discourse Treebank* using the context, word pair information, internal constituent, and dependency parsers as features. Doan et al. [29] utilize POS tags and dependency parse trees to identify conditionals based on a manually generated set of patterns from a large data set of tweets. Bui et al. [25] apply a *Logistic Regression* to detect conditionals in medical literature related to HIV. Sorgente et al. [202] combine a rule-based system with a ML model. They create a set of logical rules based on word dependencies and use these patterns to extract potential *antecedent-consequent* pairs. Finally, they use a *Bayesian* approach to discard incorrect pairs that have been produced by ambiguous patterns.

Figure 3.1 shows that many of the ML-based conditional extraction approaches manage to extract *antecedents* and *consequents* at phrase level. From sentences such as *"If A is true and B is false, then C shall occur"*, they extract the *antecedents* and *consequents* as follows: **antecedent$_1$**: *"A is true"*, **antecedent$_2$**: *"B is false"*, **consequent$_1$**: *"C shall occur"*. Consequently, the extracted text fragments are more fine-granular than the single word representations identified by the knowledge-based approaches. However, the combinatorics between the *antecedents* and *consequents* is still lost during the extraction, rendering these approaches unsuitable for our use case.

3.2.3 Deep Learning-Based Approaches

In recent years, DL-based methods achieved state of the art results in many NLP tasks, including *Text Classification* [203], *Constituency Parsing* [204], *Coreference Resolution* [205], *Information Extraction* [206] and *Named Entity Recognition* [207]. Driven by the continuous advancements of DL models and their ability to automatically extract useful features from raw text, research on conditional extraction shifted towards the usage of DL-based approaches in the last years.

Xu et al. [208] present SDP-LSTM, a neural network to classify the relation of two entities in a sentence (e.g., *Cause-Effect*, *Product-Producer*). They apply the *Shortest Dependency Path* (SDP) between two entities along with *Long Short Term Memory* (LSTM) units to detect the relationship between two entities. Ponti and Korhonen [209] use a *Feedforward Neural Network* to extract conditionals from NL. They train and evaluate their network on the *Penn Discourse Treebank* in English [210] and the *CSTNews* corpus in Brazilian Portuguese [211]. By using a set of different features, including lexical features and position features, they achieve an F_1 score of 54.5 % (*Discourse Treebank*) and 55.6 % (*CSTNews* data set). Kruengkrai et al. [212] explore how to combine *Multi-Column Convolutional Neural Networks* with background knowledge to improve the recognition of *antecedents* and *consequents*. To this end, they analyze descriptions in web texts that are somehow related to a potential *antecedent*/*consequent* candidate and train a network to judge whether the candidate is part of a conditional. Zhang et al. [213] show to improve the performance of *Graph Convolutional Networks* tailored for conditional extraction using dependency trees. Dasgupta et al. [70] use a bidirectional LSTM architecture for conditional extraction. They utilize word embeddings as features and train the network to build a causal graph illustrating the relationship of *antecedents* and *consequents*. Li et al. [71] builds on this work by adding *Flair* embeddings [214] and the self-attention mechanism to the network. Their main idea behind the usage of pre-trained embeddings is to alleviate the problem of training data insufficiency. Similarly,

Kyriakakis et al. [215] use *Transfer Learning* to detect conditionals in NL. However, we see a major problem with this paper. They train and evaluate their approaches on strongly unbalanced data sets with conditionals to non-conditionals ratios of 1:2 and 1:3 and only report the macro-Recall and macro-Precision values but not the metrics per class. Thus, it is not clear whether the classifier has a bias toward the majority class or not.

The proposed DL-based approaches show improved performance in extracting conditionals compared to the previously mentioned knowledge-based and ML-based methods. In fact, the approach proposed by Li et al. achieves an F-score of 84.4 % when evaluated on the *SemEval-2010* task 8 data set [216]. However, all described approaches are not capable of extracting conditionals in fine-grained form (see Figure 3.1).

ℹ Summary and Relation to Our Thesis:

There are a number of methods for the extraction of conditionals: rule-based systems, ML-based systems, and DL-based systems – all of which are not suitable for our use case for the following reasons:

① They are not capable of extracting conditionals in fine-grained form, which can be illustrated by the following example: *"If A is true and B is false, then C shall occur"*. Most of the mentioned approaches (see Figure 3.1) extract *antecedents* and *consequents* only on word level (i.e., **antecedent$_1$**: *"A"*, **antecedent$_2$**: *"B"*, **consequent$_1$**: *"C"*). Consequently, valuable information about the conditional is lost (e.g., the conditions of *"A"*, *"B"* and *"C"* are ignored). Others manage to extract *antecedents* and *consequents* at phrase level (i.e., **antecedent$_1$**: *"A is true"*, **antecedent$_2$**: *"B is false"*, **consequent$_1$**: *"C shall occur"*). However, these extracted text fragments are not fine-grained enough for our use cases. In order to derive test cases, *antecedents* and *consequents* must be further decomposed into variable and condition (i.e., **antecedent_variable$_1$**: *"A"*, **antecedent_condition$_1$**: *"is true"*). In addition, we need to understand the combinatorics between the *antecedents* and *consequents* and extract the relation accordingly: the exemplary requirements states that *"A"* and *"B"* are supposed to occur together before *"C"* shall occur.

② The existing DL and DL-based approaches have been trained on corpora not originating from software engineering (e.g., *BBC news* [217]) and are therefore difficult to utilize for RE purposes. Since RE documents often exhibit a specific vocabulary, we require an approach that is trained on RE data [218].

③ Neither the code nor any demos are publicly available for the existing methods. Hence, they can not be used without extensive re-implementation efforts.

We address this research gap and present our tool-supported approach `CiRA`, capable of detecting conditional statements in NL requirements (see Chapter 6) and of extracting them in fine-grained form (see Chapter 7). A live demo of `CiRA` can be accessed at `www.cira.bth.se/demo/`. Chapter 8 demonstrates the use of `CiRA` by means of an example and provides a description of its user interface.

3.3 Automated Extraction of Test Cases From Requirements

The automated extraction of test cases from requirements is an active field of research. In recent years, a number of NLP-based approaches have been introduced to assist test designers in deriving test cases. In this section, we provide an overview of these approaches and relate them to the contributions of this thesis. We build on systematic literature mapping studies conducted by Garousi et al. [55] and Ahsan et al. [219] that provide comprehensive insight into the state of the art in NLP-assisted software testing. The type of requirements from which the test cases are derived varies strongly. There is a rich body of work on automatically deriving test cases from formal (see Section 3.3.1) and semi-formal requirements (see Section 3.3.2). However, only a few approaches allow to automatically create test cases from informal requirements (see Section 3.3.3). The contribution of Chapter 10 builds upon and expands this state of the art.

3.3.1 Automatic Test Case Derivation From Formal Requirements

Definition: Formal Requirements Formal notations of requirements follow a precise syntax and semantics. Instead of expressing the expected system behavior using natural language, requirements engineers must follow a given formal specification language. The application of pre-defined languages shall aid in describing the system behavior in a precise, consistent, and unambiguous manner [220]. Examples of formal specification languages represent *Z* [221], *B* [222], *Petri nets* [223], and the *Vienna Development Method* (VDM) [224], which are based on mathematical notations.

Overview of Automated Test Case Derivation Methods Automatic test case generation methods have been proposed for different kinds of formal specification styles [225, 59]. Our description does not provide complete coverage of these methods. We do only include a sampling of the most relevant work in this section.

Sharma and Biswas [226] show how test cases can be automatically derived from courteous logic representations of requirements. Specifically, they transform a requirement like *"If persons are library members, then they can borrow a book"* into a corresponding formula written in courteous logic: `if library member(?X) and book(?Y) then borrow(?X, ?Y)`, and generate functional test cases automatically. Liu und Nakajima [59] present the *Vibration-Method* for automatic generation of test cases from model-based formal specifications. In this context, requirements are defined as functional scenarios that specify a relation between input and output. These scenarios are formalized by using the *Structured Object-Oriented Formal Language* (SOFL) [227], which can be illustrated by the following example. Let us assume we want to specify an operation called *"Purchase Ticket From Card"* (short: PTFC) that will be implemented in a ticket sales system for public transport. We allow infants to ride for free and offer students a 50 % discount on the regular fare. This operation can be defined in SOFL as follows [59]:

process PTFC(status: string, fare: nat0) actualFare: int
ext wr card: Card
pre fare ≤ card.buffer
post case status of
"Normal" → actualFare = fare and card = modify(card,buffer → card.buffer - actualFare);
"Infant" → actualFare = 0 and card = card;

"Student" → actualFare = fare * 0.5 and card = modify(card,buffer → card.buffer - actualFare); default → actualFare = -1 and card = card;

We define *Card* as a composite type that contains fields such as a *buffer* indicating the current amount of available money. The operator **ext wr** specifies the card as a writable external variable [228]. Initially, the fare and the buffer are compared. In case there is not enough money available on the card to pay the fare, the default case will be executed (see last **post** condition). No ticket can be purchased (indicated by actualFare = -1) and the card remains unchanged. Otherwise, the system shall check the status (*"Normal"*, *"Infant"*, or *"Student"*) of the customer and adapt the fare accordingly. Liu und Nakajima [59] present different approaches to convert such functional scenarios into suitable test cases automatically. In essence, they propose to apply *scenario-path coverage* meaning that a suitable test set is expected to contain both test cases that satisfy the **pre** and **post** conditions of all scenarios and test cases that check the negative paths by violating **pre** and **post** conditions.

Lee and Yannakakis [229] illustrate that there are a number of approaches concerned with test case generation from requirements modeled as finite state machines and their extended forms such as abstract state machines and state charts. A prominent example can be found in the work by Chow [230]. All developed approaches consider a system under test as a *labeled transition system*. It consists of states and labeled transitions between the states. The transition label defines the action resp. the behavior of the system (e.g., inputs and outputs). Tretmans and Brinksma [231] introduce the tool *TorX* that can be used to derive tests from formal, transition system-based specifications. It supports formal languages such as *Lotos* [232] and *Promela* [233]. Other widely used methods for deriving test cases from specifications given in the form of a finite state machine include the partial *W* method [234] and the *Unique Input Output* method [235]. Helke et al. [236] and Burton [237] explain how test cases can be generated from requirements written in *Z*. Satpathy et al. [238] discuss automatic testing of systems implemented in accordance with a formal model in *B*.

3.3.2 Automatic Test Case Derivation From Semi-Formal Requirements

Definition: Semi-Formal Requirements In the case of semi-formal requirements, the *"elements of a system and their relationships are declared formally, but the statements describing their properties are specified informally"* [220]. Examples of semi-formal notations are graphical representations such as use case diagrams and CNL. As the name indicates, a CNL builds on *Natural Language* but it is more restrictive concerning vocabulary, syntax, and/or semantics in order to reduce ambiguity and to enable automatic processing of the specification [239]. A prominent example of a CNL in *Requirements Engineering* are the EARS patterns [152]. The EARS patterns dictate a certain structure when defining requirements (e.g., *"WHILE <in a specific state> the <system name> shall <system response>"*), but allow the requirements engineer to populate the structure using natural language.

Overview of Automated Test Case Derivation Methods Carvalho et al. [240] show how to automatically generate test cases from requirements written in SysReq-CNL. They propose an approach called $NAT2TEST_{SCR}$ capable of (1) detecting whether a

requirement is expressed in SysReq-CNL, (2) creating *Software Cost Reduction* (SCR) specifications from syntactically valid requirements, and (3) generating test cases from the created SCR specifications. The test cases are created by using the *T-VEC* tool [241].

De Figueiredo et al. [242] describe how test cases can be automatically derived from requirements expressed as use cases. For this purpose, a formal behavioral model in CSP notation [243] is automatically derived from the use cases. Subsequently, they utilize a test case extraction tool [244] to generate test cases from the CSP model automatically. There are several other works dealing with the generation of test cases from use cases. For example, Barros et al. [58] present a CNL specifically designed to model use cases (called *ucsNL*) and utilize the tool *TaRGeT* [245] to derive test cases from use case scenarios written in ucsNL. Badri et al. [246] combine use case descriptions with collaboration diagrams and apply the *Collaboration Diagrams Description Language* (CDDL) to describe the semantic content of a collaboration diagram in textual form. The CDDL specifications are used for the subsequent test sequences generation process. Nebut et al. [247] associate use cases with contracts and parameters, i.e. they define the inputs to the use cases as well as pre and postconditions. Finally, test cases are generated automatically by instantiating the use case parameters. Araújo et al. [248] present an approach capable of deriving test cases from use cases expressed by a specific type of CNL: namely *CARNAUbA*. A large number of papers on the derivation of test cases from restricted use case models originate from the *Interdisciplinary Centre for Security, Reliability, and Trust* headed by Lionel Briand. Wang et al. [56, 249, 250] and Zhang et al. [251] describe how to automatically generate executable system test cases from a more structured and analyzable form of use case specifications, i.e., *Restricted Use Case Modeling* (RUCM).

Yue et al. [252] present the *Restricted Test Case Modeling* (RTCM) language and a test case generation tool (called *aToucan4Test*) that is capable of creating test cases from RTCM specifications automatically. In this context, RTCM can be understood as a specific type of CNL composed of an easy-to-use template, a set of restriction rules, and keywords targeted to the description of test case specifications. Sarmiento et al. [253] propose the *Scenarios & Lexicons* tool capable of automatically transforming semi-formal descriptions of requirements into UML activity diagrams. In this context, requirements are described by a scenario language and are structured with respect to a set of predefined entities (e.g., context, episodes). Test cases are derived from the created activity diagrams by using graph search strategies. In further work, Sarmiento et al. [254] explain how semi-formal requirements can be translated into *Petri-Net* models, which can be used as input for test scenario generation.

In response to the desire for increasing test automation, *Behavior Driven Development* (BDD) emerged in recent years [255]. BDD involves a more formalized approach to requirements, which are documented according to a specific description language (e.g., *Gherkin*). *Gherkin* suggests writing requirements in a *"Given-When-Then"* format to make them more precise and easier to translate into tests [256]. BDD advises to first transform the requirements into tests and then to successively develop each unit of the software until the tests pass. For this purpose, tools like *Cucumber* offer suitable support by automatically generating code fragments from *Gherkin* requirements [257]. They create a corresponding method skeleton for each *"Given-When-Then"* clause and thereby ensure optimal traceability between requirements and tests. Rane [258] deals with the transfor-

mation of user stories into test cases. Specifically, he focuses on the user story template *"As a [user role] I want [functionality] in order to [provide business value]"* and outlines how NLP can be used to transform user stories into activity diagrams. He applies the *Depth First Search* algorithm [259] to derive test cases from the created activity diagram.

3.3.3 Automatic Test Case Derivation From Informal Requirements

Definition: Informal Requirements Informal requirements are described in a narrative form by using *Unrestricted Natural Language*. Consequently, they can also be specified by individuals who are not experts in requirements definition since no formal rules need to be observed. However, the usage of NL is often subject to vague expressions, which can lead to divergent interpretations and, eventually, to errors in the implementation of informal requirements.

Overview of Automated Test Case Derivation Methods The need to automatically derive test cases from informal requirements has been described in early work by Sneed [105]. He presents a tool called *Text Analyzer* capable of extracting test cases from functional specifications. These specifications need to be enriched by the user with certain keywords (e.g., the term *"ACT"* is used to indicate system actors) in order to enable the tool to recognize specific requirement elements embedded in the prose text.

Masuda et al. [260] apply a set of pre-defined syntactic rules to extract conditions and actions from NL requirements. Specifically, they apply different NLP techniques (e.g., *Constituency Parsing*) to requirements and analyze the output regarding specific rules. Subsequently, they apply decision table testing to generate test cases based on the identified actions and conditions.

Dwarakanath and Sengupta [261] present *Litmus*, a tool that applies a syntactic parser called *Link Grammar* in order to analyze the structure of a NL requirement and create test cases accordingly. Goffi et al. [262] demonstrate how to create test cases for exceptional behaviors from *Javadoc* comments. First, they extract all *Javadoc* comments that are related to exceptional behaviors. Second, they translate *Javadoc* conditions into *Java* Boolean expressions (e.g., *"the comparator is locked"* is transformed to `target.isLocked()==true`). Finally, they generate test oracles in the form of assertions and embed them in test cases.

Santiago Júnior and Vijaykumar [263] present *SOLIMVA* capable of translating NL requirements into state charts used for the eventual test case generation. Verma and Beg [264] as well as Kulkarni and Joglekar [52] describe a similar approach for translating informal requirements into knowledge representation graphs. They use a boundary value analysis for the automatic generation of test cases from the created knowledge graphs.

Ansari et al. [265] propose an NLP system capable of extracting test cases from conjunctive statements containing *"if"* and *"then"* keywords. However, the description of the system contained in their paper is very rudimentary, making it difficult to understand the inner workings of the NLP pipeline.

Boddu et al. [266] present a requirements analysis tool called *REquirements to Testing in a NAtural way* (RETNA). First, RETNA classifies NL requirements according to their type and complexity. Second, the requirements are translated to an intermediate predicate-argument structure [267] and later to a discourse representation structure [268]. Based on user interaction, the translated requirements are refined and ultimately transformed into a state machine, from which test cases are derived automatically.

ℹ Summary and Relation to Our Thesis:

NLP-assisted software testing is an active field of research. There are a number of approaches that support practitioners in deriving test cases from formal, semi-formal, and informal requirements. In this thesis, we focus on automatic acceptance test creation from informal requirements - the most common notation style of requirements in practice. The approaches presented in Section 3.3.1 and Section 3.3.2 require the compliance of a rigid formalism and are therefore not suitable for the implementation of our use case. Similarly, the approaches presented for deriving test cases from informal requirements are also not suited due to the following drawbacks:

① Most importantly, the presented approaches do not ensure that only the minimal number of required test cases is created. However, this is crucial because practitioners need assistance in striking a balance between full test coverage and the number of required test cases (see Section 1.2).

② They show poor performance when evaluated on unseen real-world data. Specifically, they are not robust against grammatical errors and fail to process words that are not yet part of their training vocabulary.

③ Partly, they require manual work such as the creation of a dictionary [263] defining the application domain in which the approach will be used or they rely on user interaction for requirements refinement [266].

④ Some authors present both their approach and a corresponding tool in their paper, but do not make it publicly available. Hence, the approaches are not immediately usable for practitioners.

We address this research gap and present an approach capable of deriving acceptance tests from conditional statements in NL requirements automatically. We use `CiRA` to extract conditionals in fine-grained form and then translate them into a CEG. The usage of a CEG enables us to automate the combinatorial design of test cases and ensure that they are minimal. Our case study proves that our approach is able to automatically create a significant amount of relevant (known and new) test cases (see Chapter 10). In our setting, our approach automatically created 71.8 % of the 578 manually created test cases. Additionally, it identified 80 relevant test cases that were missed in manual test design.

Part I

Understanding Conditionals in Requirements Artifacts

Common Thread As outlined in Section 1.3, we lack knowledge on the notion of conditional statements in requirements artifacts. Specifically, we do not know how prevalent conditionals are in RE artifacts, as well as in which form and complexity they occur. This hinders the development of approaches for their extraction since we do not understand which capabilities a suitable extraction approach needs to possess. For example, does the approach need to be able to handle conjunctive and disjunctive events or do conditionals in RE artifacts only contain single *antecedent-consequent* relationships? In addition, we still lack knowledge on how conditionals are interpreted in practice, and how they should be formalized accordingly. For example, do practitioners perceive an *antecedent* as only *sufficient* or also *necessary* for a *consequent*? This knowledge is indispensable to build a suitable approach for conditional extraction since conditionals must be associated with formal semantics to automatically process them and utilize their embedded logical knowledge. In this part of the thesis, we provide an answer to these questions. We present two studies that help the reader build an understanding of conditionals in RE artifacts. In the first study (see Chapter 4), we analyze the extent, form, and complexity of conditionals in requirements rooted in 14,983 sentences and emerging from 53 requirement documents. In the second study (see Chapter 5), we study how 104 RE practitioners interpret 12 different conditional clauses in requirements. At the end of each study, we discuss the implications of our results for the development of an approach for automated conditional extraction from RE artifacts.

Preliminaries To understand this part, it is first necessary to comprehend the theoretical foundations described in Section 2.1. The reader must understand the concept of conditional statements and be aware of the different forms in which conditionals can occur. In addition, the distinction between the different logical levels of interpretation (*Necessity* and *Temporality*) is elementary to follow the second study.

Chapter 4

Empirical Study on Prevalence, Form, and Complexity of Conditionals in Requirements Artifacts

Common Thread We conduct an exploratory case study to address the first problem of this thesis, namely *"the missing understanding of the notion of conditionals in RE artifacts"* (see Section 1.3). Specifically, we investigate 14,983 sentences from 53 requirements documents originating from 18 different domains and shed light on the prevalence, form, and complexity of conditionals in requirements. We involve six annotators and annotate the sentences in our data set with respect to nine categories (e.g., *explicit / implicit* conditionals, *marked / unmarked* conditionals). We verify the reliability of our annotations by calculating Gwet's AC1 measure. Across all categories, we achieve a mean value of above 0.8, which indicates a nearly perfect agreement.

Contribution Our study demonstrates that conditional statements matter in RE artifacts. About 28 % of the analyzed sentences specify a dependency between an *antecedent* and a *consequent*. We found that the majority (56 %) of conditional sentences contained in requirement documents express an *enable* relationship between certain events. Further, our study shows that most conditionals in RE artifacts are *explicit* and occur in *marked* form. Single *antecedents* and *consequents* occur significantly more often than multiple *antecedents* and *consequents*. However, we also found that conditionals in RE artifacts may include up to three *antecedents* and two *consequents*. Hence, for the conditional extraction approaches to be applicable in practice, they must be capable of understanding conjunctions, disjunctions and negations in the sentences to fully capture the relationships between *antecedents* and *consequents*.

Related Publications This chapter is taken, directly or with minor modifications, from previous publications [1, 4, 7].

4.1 Research Objective

Reliable knowledge about the distribution of conditionals is a necessary precondition to develop efficient approaches for the automated extraction of conditional statements. However, empirical evidence on conditionals in requirements artifacts is presently still weak. We address this research gap and analyze the prevalence, form, and complexity of conditional statements in requirements artifacts. Based on the terminology introduced in Section 2.1, we investigate the following research questions (RQ):

- **RQ 1:** To which degree do conditionals occur in requirement documents?
- **RQ 2:** How often do the relations *cause*, *enable* and *prevent* occur?
- **RQ 3:** In which form do conditional statements occur in requirement documents?
  - **RQ 3a:** How often do *marked* and *unmarked* conditionals occur?
  - **RQ 3b:** How often do *explicit* and *implicit* conditionals occur?
  - **RQ 3c:** Which cue phrases are used? Are they mainly *ambiguous* or *non-ambiguous*?
- **RQ 4:** At which complexity do conditionals statements occur in requirement documents?
  - **RQ 4a:** How often do multiple *antecedents* occur?
  - **RQ 4b:** How often do multiple *consequents* occur?
  - **RQ 4c:** How often do two sentence conditionals occur?
  - **RQ 4d:** How often do *event chains* occur?
- **RQ 5:** Is the distribution of labels in all categories domain-independent?

To answer our research questions, we perform a case study according to the guidelines of Runeson and Höst [269]. Based on the classification of Robson [270], our case study is exploratory as we seek for new insights into conditionals in requirement documents.

4.2 Study Design

In this section, we characterize our study objects and describe how we conducted the case study to answer our research questions. Specifically, we outline the process of annotating the study objects and specify the applied procedures to ensure the validity of our annotations.

Study Objects To obtain evidence on the extent to which conditional statements are used in requirements artifacts in practice, we had to generate a large and representative collection of artifacts. We considered data sets as eligible for our case study based on three criteria: 1) the data set shall contain requirements artifacts that are/were used in practice, 2) the data set shall not be domain-specific, but rather contain artifacts from different domains, and 3) the documents shall originate from a time frame of at least 10 years. Following these criteria ensures that our analysis is not restricted to a single year

Table 4.1: Overview of Collected Requirement Documents.

Website	Search Terms	# Docs
google.com	requirements document (pdf) OR requirements specification (pdf)	124
ntrs.nasa.gov	requirements document	11
sci.esa.int	requirements document	7
www.eurocontrol.int	requirements	50
www.everyspec.com	requirements OR system requirements	178
www.etsi.org/standards	most recent	93

or domain, but rather allows for a comprehensive and generalizable view on conditionals in requirements. The *PUblic REquirements data set* (PURE) [218] is a first contribution in this direction, however, it is not suitable for our purposes as it is not clear from which domains and years the contained requirements documents originate. Hence, we initiated the creation of a new large gold standard corpus of requirements [2]. The corpus is not only intended to be used for our study purposes but should also serve as the basis for further studies in the RE community. For example, we welcome fellow researchers to use the data set as a benchmark for different RE relevant NLP tasks such as requirements classification.

We collected publicly available requirements specifications by means of a web search. We queried *Google* and libraries as *Everyspec* to retrieve documents from different domains. Table 4.1 summarizes the searched websites and the applied search terms. We only considered documents that are in PDF format, have at least 10 pages, are written in English, and do contain requirements. To verify the latter, we conducted a brief manual review of each document. Our search led to the identification of 463 requirement documents. The shortest document has 10 pages, while the longest contains 2822 pages. On average, a document has 88 pages. In order to make the data included in the documents usable for further analysis, we have to extract complete sentences. Simply extracting the lines and analyzing incomplete sentences is not reasonable as essential phrases of the sentences are neglected. Additionally, we need to clean up the data, i.e. we need to remove lines that are only used for structure purposes (e.g. headings) and do not contain RE relevant content. To this end, we performed the following steps:

① Extract raw text lines from the PDF file using the *pdfminer Python* library.

② Preprocess the resulting lines by removing leading and trailing white spaces and compressing a group of consecutive white spaces to one.

③ Filter out lines that contain clearly no content. A line is filtered if at least one of the following criteria is satisfied:

a) The line starts with *"Figure"* or *"Table"*.
b) The line starts with a page character or *"Chapter"*.
c) The line contains less than 50 characters and does not end with *"."*, *"?"* or *"!"*.
d) The line contains a group of at least 4 consecutive *"."* characters (deletion of entries in table of content).

④ Delete enumeration marks like *"a)"*.

⑤ Build paragraphs by splitting the text at empty lines and combining consecutive lines.

⑥ Split the paragraphs into sentences.

⑦ Manually improve the sentence splitting by combining two consecutive sentences if the first sentence ends with *"e.g."* or *"i.e."*.

⑧ Remove the prefix *"Note"* if it is not followed by *"that"*.

After preprocessing, our data set contains 212,186 complete sentences.[1] To the best of our knowledge, this data set is currently the most extensive collection of requirements available to the research community. We randomly selected 53 documents from the data set for our analysis of the prevalence of conditionals in RE artifacts. Hence, our study focuses on 14,983 sentences from 18 different domains (see Figure 4.1).

Model the Phenomenon In order to answer our research questions, we need to annotate the sentences in our data set with respect to certain categories (e.g. *explicit* or *implicit* conditionals). According to Pustejovsky and Stubbs [271], the first step in each annotation process is to *"model the phenomenon"* that needs to be annotated. Specifically, it should be defined as a model M that consists of a vocabulary T, the relations R between the terms as well as the interpretations I of terms. RQ 1 can be understood as a binary annotation problem, which can be modeled as:

- **T:** {sentence, Conditional Present, Conditional Not Present}
- **R:** {sentence ::= Conditional Present | Conditional Not Present}
- **I:** {Conditional Present = A sentence contains a conditional statement if it specifies a relation between at least two events e_1 and e_2, where e_1 leads to the occurrence of e_2; Conditional Not Present = A sentence does not contain a conditional if it describes a state that is independent on any events}

Modeling an annotation problem has two advantages: It contributes to a clear definition of the research problem and can be used as a guide for the annotators to explain the meaning of the labels. We have modeled each RQ and discussed it with the annotators. In addition to interpretation I, we have also provided an example for each label to avoid misunderstandings. After modeling all RQs, the following eight categories emerged, according to which we annotated our data set: Conditional Present, Explicit, Marked, Single Sentence, Single Antecedent, Single Consequent, Event Chain, and Relationship. We refer to all categories except Conditional Present as *dependent* categories, as they are dependent on the Conditional Present label. To answer RQ 6, we perform a stratified analysis for each of the aforementioned categories using the domains as strata. Due to the imbalance of the data set with respect to the domains the requirements sentences originate from, we formulate the following null hypothesis for each category X: *"sentences from different domains have the same distribution of values in category X"*.

1 Available at `https://figshare.com/s/725309c06b9dc82aa4a1`. Due to the terms of use of some sources, we can only share the URLs of the collected documents. We attached a script to download the data set automatically.

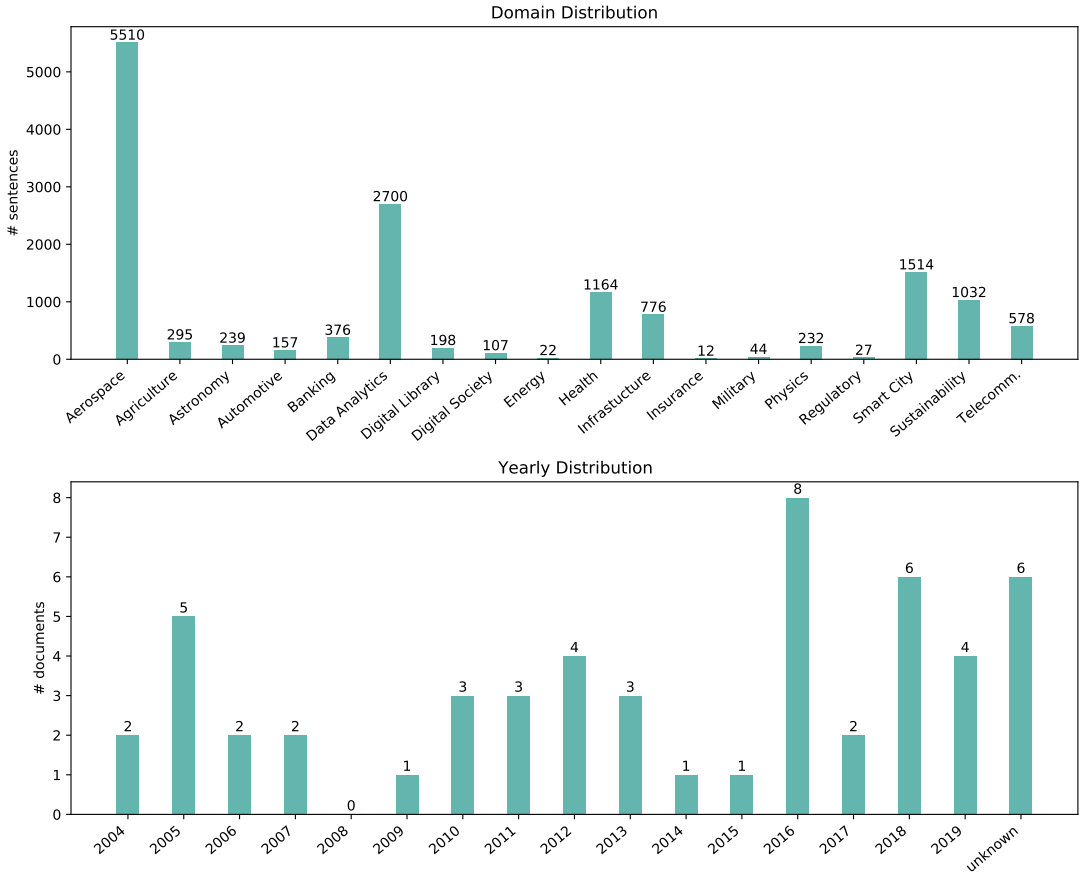


Figure 4.1: Descriptive Statistics of Our Data Set. The Upper Graph Shows the Number of Sentences per Domain. The Lower Graph Depicts the Year of Creation per Document.

Annotation Environment We developed our own annotation platform tailored to our research questions.[2] Contrary to other annotation platforms [272] which only show single sentences to the annotators, we also show the predecessor and successor of each sentence. This is required to determine whether the conditional extends over one sentence or across multiple ones (see RQ 5c). For the binary annotation problems (see RQ 1, RQ 4a, RQ 4b, RQ 5a - d), we provide two labels for each category. Cue phrases present in the sentence can either be selected by the annotator from a list of already labeled cue phrases or new cue phrases can be added using a text input field (see RQ 4c). Since RQ 2 and RQ 3 are ternary annotation problems, the platform provides three labels for these categories.

Annotation Guideline To ensure a common understanding both of conditionals in general and of the respective categories, we conducted a workshop with all annotators prior to the labeling process. The results of the workshop were recorded in the form of an annotation guideline.[3] All annotators were instructed to comply with all of the annotation rules. One important, initially counter-intuitive instruction was to not entirely depend on the occurrence of cue phrases, as this approach is prone to introducing too many False Positives. Rather than focusing on lexical or syntactic attributes, the annotation process has to be initiated by fully reading the sentence and comprehending it on a semantic level. The impact of this becomes evident when considering some examples: requirements like *"If the gaseous nitrogen supply is connected to the ECS duct system, ECS shall include the capability of monitoring the oxygen content in the ducting"* are easy to classify since the conditional is indicated by the cue phrase *"if"* and due to the explicit phrasing of both the *antecedent* and the *consequent*. Requirements containing a relative clause like *"Any items or issues which will limit the options available to the platform developers should be described"* are more difficult to correctly classify due to the lack of cue phrases. The semantically equivalent paraphrase *"If an item or issue will limit the options available to the platform developers, the item or issue should be described"* reveals the conditional statement contained by the requirement.

Annotation Validity To verify the reliability of our annotations, we calculated the inter-annotator agreement. We assigned 3,000 sentences to each annotator, of which 2,500 are unique and 500 overlapping. Based on the overlapping sentences, we calculated the Cohen's Kappa [273] measure to evaluate how well the annotators can make the same annotation decision for a given category. We chose Cohen's Kappa since it is widely used for assessing inter-rater reliability [274]. However, a number of statistical problems are known to exist with this measure [275]. In case of a high imbalance of ratings, Cohen's Kappa is low and indicates poor inter-rater reliability even if there is a high agreement between the raters (Kappa paradox [276]). Thus, Cohen's Kappa is not meaningful in such scenarios. Consequently, studies [277] suggest that Cohen's Kappa should always be reported together with the percentage of agreement and other paradox resistant measures (e.g. Gwet's AC1 measure [278]) in order to make a valid statement about the inter-rater reliability. We involved six annotators in the creation of the corpus and assessed the inter-rater reliability on the basis of 3,000 overlapping sentences, which represents about 20 %

2 The platform can be accessed at `clabel.diptsrv003.bth.se`.

3 The annotation guideline can be found in our replication package: `https://doi.org/10.5281/zenodo.5596668`.

Table 4.2: Inter-Annotator Agreement Statistics per Category. The Category Relationship Was Jointly Labeled and Therefore Does Not Require a Reliability Assessment.

		Conditional Present		Explicit		Marked		Single Sentence		Single Antecedent		Single Consequent		Event Chain		**avg.**
		0	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	
Confusion	**0**	2034	193	24	25	1	22	12	8	41	77	63	72	450	27	
Matrix	**1**	274	499	39	411	12	464	17	462	43	338	46	318	13	9	
Agreement		84.4 %		87.2 %		93.1 %		95.0 %		76.0 %		76.4 %		92.0 %		86.3 %
Cohen's Kappa		0.579		0.358		0.023		0.464		0.261		0.362		0.27		0.331
Gwet's AC1		0.753		0.84		0.926		0.945		0.645		0.625		0.91		0.806

of the total data set. We calculated all measures (see Table 4.2) using the cloud-based version of *AgreeStat* [279]. Cohen's Kappa and Gwet's AC1 can both be interpreted using the taxonomy developed by Landis and Koch [280]: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

Table 4.2 demonstrates that the inter-rater agreement of our annotation process is reliable. Across all categories, an average percentage of agreement of 86 % was achieved. Except for the categories Single Antecedent and Single Consequent, all categories show a percentage of agreement of at least 84 %. We hypothesize that the slightly lower value of 76 % for these two categories is caused by the fact that in some cases the annotators interpret the *antecedents* and *consequents* with different granularity (e.g., annotators might break some *antecedents* and *consequents* down into several sub *antecedents* and *consequents*, while some do not). Hence, the annotations differ slightly. The Kappa paradox is particularly evident for the categories Marked and Event Chain. Despite a high agreement of over 90 %, Cohen's Kappa yields a very low value, which *"paradoxically"* suggests almost no or only fair agreement. A more meaningful assessment is provided by Gwet's AC1 as it did not fail in the case of prevalence and remains close to the percentage of agreement. Across all categories, the mean value is above 0.8, which indicates a nearly perfect agreement. Therefore, we assess our labeled data set as reliable and suitable for further analysis.

Data Analysis RQ 1-5 are answered by providing descriptive statistics of the distribution of labels for each category. For RQ 6, inferential statistics are applied. Since the hypotheses formulated for each category aim to investigate the independence between the association of a requirement to a specific domain and the distribution in the respective category, a statistical hypothesis test for independence can be used. As both the independent variable (the domain) and the dependent variable (the respective category) are categorical, the Chi-squared test will be used. The category Conditional Present is tested with respect to the full annotated data set. All dependent categories are tested on the samples that contain conditionals since only sentences including conditionals are annotated in the other categories. For all tests, only domains with at least 100 sentences were selected as eligible strata to confine the hypothesis tests to sufficiently represented domains. This threshold was introduced to RQ 6 to avoid the noise of underrepresented domains. Since the data set was aggregated in RQ 1-5, this change is only necessary for

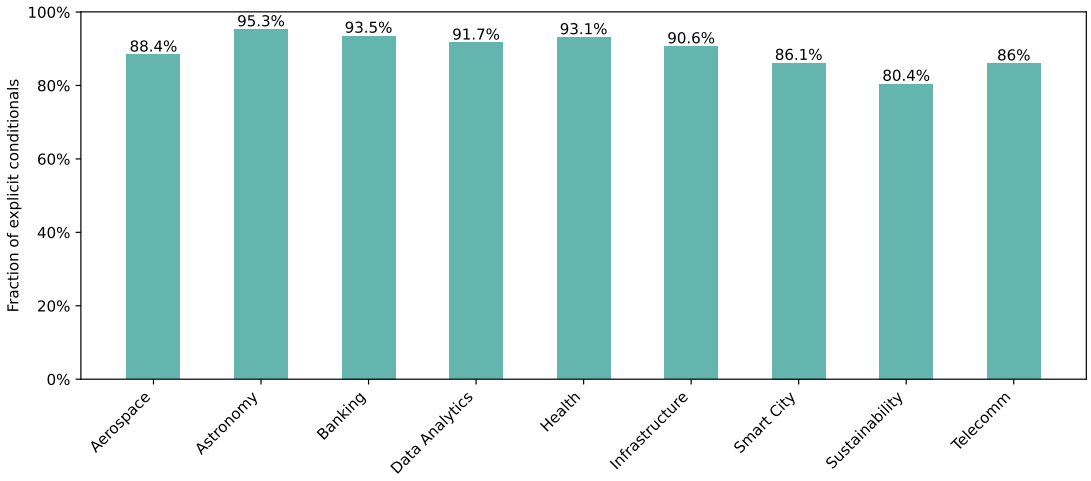


Figure 4.2: Percentage of Sentences Labeled as Explicit Within Domains Containing at Least 100 Conditionals.

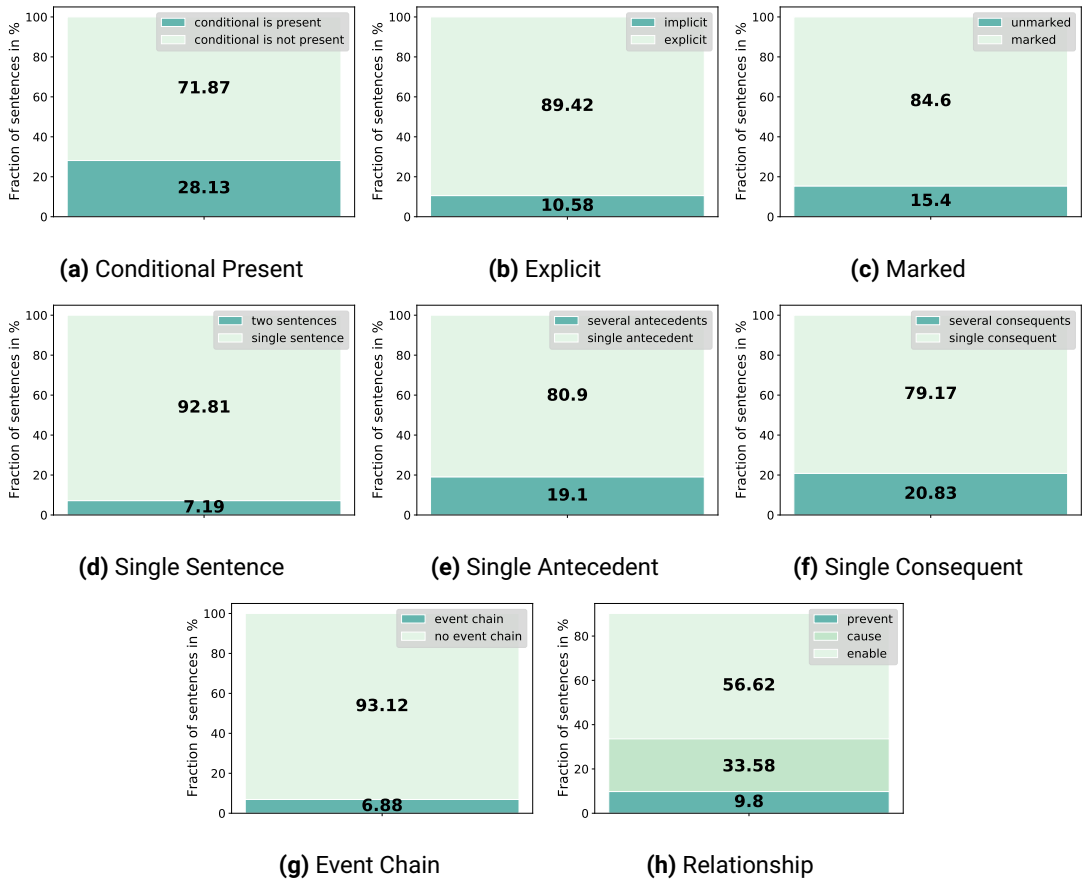
RQ 6. Using the subset of domains as strata implies the degree of freedom of the Chi-squared tests exceeding 2, hence the risk of the multiple comparison problem, i.e., the likelihood for a Type I error in rejecting null hypotheses, arises [281]. For example, when evaluating the null-hypothesis of independence for the dichotomous category Explicit, considering the nine eligible domains with more than 100 sentences including conditionals yields a degree of freedom of 8, as it is calculated as follows [282] (considering that the number of rows is 2 for dichotomous variables):

$$dof = (\text{number of rows} - 1) * (\text{number of columns} - 1) = (2-1) * (9-1) = 8 \tag{4.1}$$

The p-value of the Chi-squared test of this hypothesis is 0.000036, far below the significance level $\alpha = 0.05$, even though the relative number of values in the category Explicit among the eligible domains suggests an equal distribution and therefore independence of the domain, as seen in Figure 4.2. Hence, instead of reporting the in this case not meaningful p-value of the Chi-square hypothesis test we perform a Bonferroni correction [281] on the significance level and perform the Chi-squared test in each category for each domain against the sum of all samples outside of the domain, as applied in similar scenarios [283]. Applying the Bonferroni correction to the significance level based on the following formula [281] yields a significance level that counteracts the large degree of freedom m and reduces the likelihood of Type I errors when refuting null hypotheses:

$$p_c = \frac{\alpha}{m} = \frac{0.5}{8} = 0.00625 \tag{4.2}$$

The previously calculated p-value for the Chi-square test of independence considering all domains still suggests to reject the null hypothesis. Hence, a post-hoc test similar to [283], where each domain is compared to the sum of all other domains, is applied to reveal, that only the null-hypothesis for the domain *sustainability* can be refuted with a p-value of $0.0001 < 0.00625$, which aligns with Figure 4.2. This procedure is applied to all hypotheses of RQ 6.



(a) Conditional Present **(b)** Explicit **(c)** Marked

(d) Single Sentence **(e)** Single Antecedent **(f)** Single Consequent

(g) Event Chain **(h)** Relationship

Figure 4.3: Annotation Results per Category. The Y Axis of the Bar Plot for the Category Conditional Present Refers to the Total Number of Analyzed Sentences. The Other Bar Plots Are Only Related to the Sentences That Contain a Conditional.

4.3 Study Results

Figure 4.3 presents the analysis results for each labeled category. When interpreting the values, it is important to note that we analyze entire requirement documents in our study. Consequently, our data set contains records with different contents, which do not necessarily represent all functional requirements. For example, requirement documents also contain non-functional requirements, phrases for content structuring, purpose statements, etc. Hence, the results of our analysis do not only refer to functional requirements but in general to the content of requirement documents.

Answer to RQ1 Figure 4.3 highlights that conditional statements occur in requirement documents. About 28 % of the analyzed sentences contain a conditional. It can therefore be concluded that conditionals are a major linguistic element of requirement documents since almost one-third of all sentences describe a dependence between an *antecedent* and *consequent*.

Side Note: On the Prevalence of Conditionals in Agile RE Artifacts

In a previous study [1], we analyzed 961 user stories describing the functionality of two *Business Information Systems* (BIS) being in production at *Allianz Deutschland*. Specifically, we studied the contained acceptance criteria and examine for recurring patterns in describing system behavior. This allowed us to detect different patterns in the formulation of acceptance criteria. Our analysis revealed that the system behavior is described recurrently using three different patterns:

① *State Description*: This pattern describes an expected state of the system associated with the fulfillment of the user story. Example: *"A list of existing insurance policies for the selected user profile is shown in the overview."*

② *Antecedent-Consequent-Relationship*: This pattern describes the expected system behavior (*consequent*) in relation to certain system inputs (*antecedents*). The acceptance criterion thus defines the functionality on the occurrence of certain *antecedents* and vice versa. Example: *"Discount applies only to drivers who are at least 23 years old and show a no-claims class of 4, or do not book a comfort package."*

③ *Process Flows*: Similar to the previous pattern, the third pattern defines functionality based on specific events. However, not only their occurrence but also their chronological order plays a major role. Example: *"After selecting a contract type, the data is transmitted to the server, and a new query is issued to the user asking whether further persons should be included in the contract. Then a toggle appears on whether the contract documents should be sent by email or post."*

We found that the second pattern is the major category (BIS 1: 46 % and BIS 2: 52 %). In about 31 % of user stories describing BIS 1, the first pattern is used, while the third pattern is used least (23 %). Regarding BIS 2, the first pattern is applied in 35 % and the third pattern in 13 % of all user stories. Hence, our

study demonstrates that conditionals are also prevalent in agile RE artifacts. This finding complements the results of the study described in the present chapter and contributes to a holistic view of the occurrence of conditionals in RE artifacts.

Answer to RQ2 The majority (56 %) of conditionals contained in requirement documents express an *enable* relationship between certain events. Only about 10 % of the conditionals indicate a *prevent* relationship. *Cause* relationships are found in about 34 % of the annotated data.

Answer to RQ3a Figure 4.3 shows that the majority of conditionals contain one or more cue phrases to indicate the relationship between certain events. *Unmarked* conditionals occur only in about 15 % of the analyzed sentences.

Answer to RQ3b Most conditionals are *explicit*, i.e. they contain information about both the *antecedent* and the *consequent*. Only about 10 % of conditionals in the investigated requirements documents are *implicit*.

Answer to RQ3c All cue phrases used to indicate conditionals statements in the investigated requirements artifacts are listed in Table 4.3. The left side of the table shows the cue phrases ordered by word group. On the right side, all verbs used to express conditionals are listed. The verbs are further ordered according to whether they express a *cause*, *enable*, or *prevent* relationship. To assess the ambiguity of a cue phrase x, we formulate a binary classification task: consider all sentences as the sample space. The conditionals of that sample space represent the relevant elements. The precision of cue phrase x as a selection criterion for conditionals is the conditional probability, that a sentence from the sample space contains a conditional given that it contains cue phrase x, and hence reflects the ambiguity of the cue phrase:

$$\Pr(\text{sentence contains conditional} \mid \text{sentence contains x}) = \frac{\Pr(\text{sentence contains conditional} \cap \text{sentence contains x})}{\Pr(\text{sentence contains x})} \tag{4.3}$$

A high precision value indicates a *non-ambiguous* cue phrase, i.e., the occurrence of the cue phrase in a sentence is a strong indicator for the sentence containing a conditional, while low values indicate strongly *ambiguous* cue phrases. Table 4.3 demonstrates that a number of different cue phrases are used to express conditionals in requirement documents. Not surprisingly, cue phrases like *"if"*, *"because"* and *"therefore"* show precision values of more than 90 %. However, there is a variety of cue phrases that indicate conditionals in some sentences but also occur in other contexts. This is especially evident in the case of pronouns. Relative sentences can indicate conditionals, but not in every case, which is reflected by the low precision value of for example *"which"*. A similar pattern emerges with regard to the used verbs. Only a few verbs (e.g., *"leads to, degrade"*, and *"enhance"*) show a high precision value. Consequently, the majority of used pronouns and verbs do not necessarily indicate a conditional if they are present in a sentence.

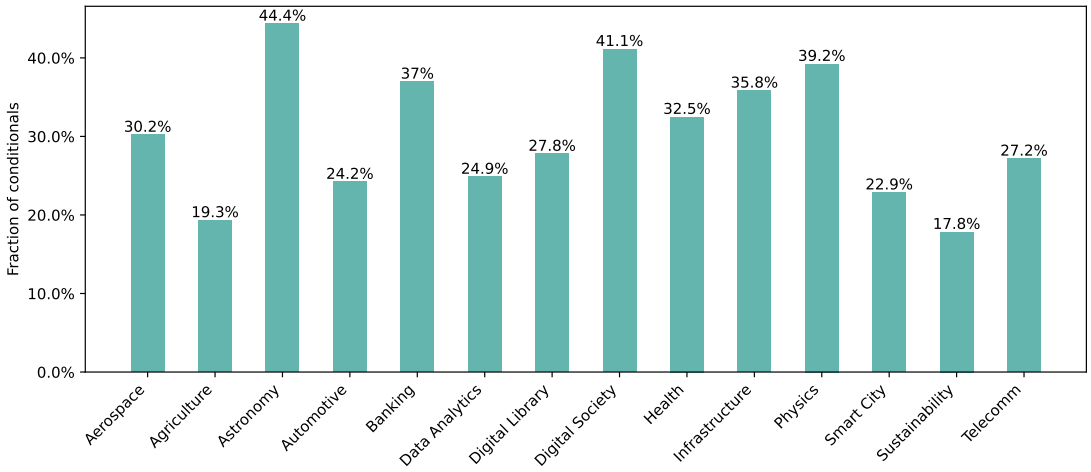


Figure 4.4: Distribution of Conditional Statements Among Domains.

Answer to RQ 4a Figure 4.3 illustrates that a conditional in requirement documents often includes only a single *antecedent*. Multiple *antecedents* occur in only 19.1 % of analyzed conditionals. The exact number of *antecedents* was not documented during the annotation process. However, the participating annotators reported consistently that in the case of complex conditional statements, two to three *antecedents* were usually included. More than three *antecedents* were rare.

Answer to RQ4b Interestingly, the distribution of *consequents* is similar to that of *antecedents*. Likewise, single *consequents* occur significantly more often than multiple *consequents*. According to the annotators, the number of *consequents* in case of complex conditionals is limited to two *consequents*. Three or more *consequents* occur rarely.

Answer to RQ4c Most conditionals can be found in single sentences. Relations where *antecedent* and *consequent* are distributed over several sentences occur only in about 7 % of the analyzed data. The annotators reported that most often the cue phrase *"therefore"* was used to express two-sentence conditionals.

Answer to RQ4d Figure 4.3 shows that *event chains* are rarely used in requirement documents. Most conditionals contain isolated relations between *antecedent* and *consequent* and only a few *event chains*.

Answer to RQ 5 Figure 4.4 visualizes the distribution of conditionals among all domains which are represented with more than 100 sentences. As the percentage of conditionals ranges from 17.8 % up to 44.4 %, we can assume that conditional statements are indeed a phenomenon occurring in all eligible domains. The Chi-squared test reported in Table 4.4 suggests rejecting the null hypothesis for domain-independence for 10 out of 14 eligible domains considering the Bonferroni-corrected significance level. We can conclude that conditionals are a phenomenon observable independent of the domain from which requirements originate, but the extent to which conditionals occur differs with statistical significance.

Table 4.3: Overview of Cue Phrases Used to Indicate Conditionals in Requirement Documents. **Bold** Precision Values Highlight Non-Ambiguous Phrases That Mostly Indicated Conditionals: $\Pr(\text{Conditional is present} \mid \text{X is present in sentence}) \geq 0.8$.

Type	Phrase	Conditional Present	Conditional Not Present	Precision	Type	Phrase	Conditional Present	Conditional Not Present	Precision
conjunctions	if	387	41	**0.90**	Cause	force(s/ed)	21	18	0.54
	as	607	1313	0.32		cause(s/ed)	32	10	0.76
	because	78	7	**0.92**		lead(s) to	5	0	**1.00**
	but	100	204	0.33		reduce(s/ed)	48	28	0.63
	in order to	141	33	**0.81**		minimize(s/ed)	28	11	0.72
	so (that)	88	86	0.51		affect(s/ed)	13	19	0.41
	unless	23	4	**0.85**		maximize(s/ed)	11	5	0.69
	while	71	90	0.44		eliminate(s/ed)	8	11	0.42
	once	48	15	0.76		result(s/ed) in	50	43	0.54
	except	9	5	0.64		increase(s/ed)	49	34	0.59
	as long as	12	1	**0.92**		decrease(s/ed)	5	8	0.38
adverbs	therefore	61	6	**0.91**		impact(s)	37	68	0.35
	when	331	64	**0.84**		degrade(s/ed)	11	2	**0.85**
	whenever	10	0	**1.00**		introduce(s/ed)	11	12	0.48
	hence	21	9	0.70		enforce(s/ed)	2	1	0.67
	where	213	150	0.59		trigger(s/ed)	11	7	0.61
	then	111	70	0.61		imply	7	14	0.33
	since	65	32	0.67		attain(s/ed)	3	13	0.18
	consequently	2	6	0.25		create(s/ed)	39	88	0.30
	wherever	5	2	0.71		impose(s/ed)	7	13	0.35
	rather	16	30	0.35		perform(s/ed)	26	60	0.30
	to this/that end	12	0	**1.00**	Enable	depend(s) on	28	21	0.57
	thus	66	17	**0.80**		require(s/ed)	316	262	0.55
	for this reason	7	3	0.70		allow(s/ed)	187	130	0.59
	due to	91	26	0.78		need(s/ed)	98	162	0.38
	thereby	4	2	0.67		necessitate(s/ed)	7	2	0.78
	as a result	11	4	0.73		facilitate(s/ed)	29	28	0.51
	for this purpose	1	2	0.33		enhance(s/ed)	16	4	**0.80**
pronouns	which	277	608	0.31		ensure(s/ed)	145	66	0.69
	who	28	52	0.35		achieve(s/ed)	30	24	0.56
	that	732	1178	0.38		support(s/ed)	128	301	0.30
	whose	16	11	0.59		enable(s/ed)	75	36	0.68
adjectives	only	127	126	0.50		permit(s/ed)	10	13	0.43
	prior to	26	20	0.57		rely on	3	5	0.38
	imperative	1	3	0.25		measure(s/ed)	99	247	0.28
	necessary (to)	36	19	0.65		provide(s/ed)	75	125	0.37
	given	73	140	0.34		get	13	23	0.36
	following	53	175	0.23		meet	42	34	0.55
preposition	for	1209	2753	0.31	Prevent	hinder(s/ed)	1	1	0.50
	during	327	137	0.70		prevent(s/ed)	38	17	0.69
	after	133	57	0.70		avoid(s/ed)	14	23	0.38
	by	506	1171	0.30		mitigate(s/ed)	3	8	0.27
	with	680	1554	0.30					
	in the course of	2	1	0.67					
	through	114	204	0.36					
	as part of	19	51	0.27					
	in this case	18	3	**0.86**					
	before	54	27	0.67					
	until	33	11	0.75					
	upon	25	48	0.34					
	in case of	30	7	**0.81**					
	in both cases	1	0	**1.00**					
	in the event of	15	2	**0.88**					
	in response to	6	7	0.46					
	in the absence of	8	1	**0.89**					
	within	150	315	0.32					
	as far as	4	5	0.44					
	according to	21	54	0.28					
	around	25	41	0.37					
	from	370	990	0.27					
	based on	56	175	0.24					

Table 4.4: Bonferroni-Corrected Chi-Squared Tests of Independence From the Domain. Cells Prefixed With * Indicate a Category, Where the Distribution of the Given Domain Differs Significantly From the Sample.

Domain	Conditional Present	Explicit	Marked	Single Antecedent	Single Consequent	Event Chain	Single Sentence	Relationship
p_c	3.8E-03	6.3E-03	6.3E-03	6.3E-03	6.3E-03	6.3E-03	6.3E-03	3.1E-03
Aerospace	*8.0E-05	1.5E-01	8.2E-03	9.8E-02	6.4E-03	7.2E-02	6.3E-01	3.1E-02
Agriculture	*7.0E-04	(domain contained less than 100 conditionals)						
Astronomy	*4.1E-08	6.2E-02	1.0E-02	8.9E-01	2.7E-01	1.1E-01	2.4E-01	3.0E-01
Automotive	2.9E-01	(domain contained less than 100 conditionals)						
Banking	*1.9E-04	1.3E-01	5.3E-01	9.0E-02	2.8E-02	5.3E-01	2.6E-01	4.4E-01
Data Analytics	*1.4E-05	3.4E-02	1.3E-02	2.3E-02	*3.7E-03	1.3E-01	3.9E-01	5.9E-01
Digital Library	9.3E-01	(domain contained less than 100 conditionals)						
Digital Society	4.4E-03	(domain contained less than 100 conditionals)						
Health	*9.4E-04	1.4E-02	5.7E-01	7.9E-02	6.8E-01	6.3E-01	7.7E-01	1.8E-01
Infrastructure	*2.1E-06	5.0E-01	3.2E-01	4.1E-01	1.6E-01	5.0E-01	6.8E-01	*6.7E-05
Physics	*2.1E-04	(domain contained less than 100 conditionals)						
Smart City	*8.4E-07	5.9E-02	*2.0E-05	1.3E-02	3.5E-01	3.2E-01	*2.3E-03	3.9E-01
Sustainability	*1.4E-14	*1.2E-04	*2.3E-04	8.7E-01	5.4E-01	1.4E-02	*5.7E-03	1.9E-01
Telecomm	5.7E-01	2.2E-01	7.3E-01	5.2E-01	3.1E-01	1.8E-02	3.5E-02	7.1E-01

For all dependent categories, the domains *Aerospace*, *Astronomy*, *Banking*, *Data Analytics*, *Health*, *Infrastructure*, *Smart City*, *Sustainability*, and *Telecomm* are eligible for consideration as they contain more than 100 sentences that include a conditional. On the right side of Table 4.4 each cell contains the p-value for a Chi-squared test comparing the distribution of the given domain to the rest of the sample. Where the p-value for a given domain and category is lower than the Bonferroni-corrected significance level (denoted for each category as p_c), the cell is prefixed with an asterisk. The Chi-squared test of independence does not suggest to reject the null hypothesis for the categories Single Antecedent and Event Chain, but the distribution of 2 out of the eligible 9 domains in the category Marked and Single Sentence are significantly different. We can conclude that the distribution of values in all categories is domain-independent to a certain degree: while the complexity of conditionals is mostly domain-independent, the distribution of marked conditionals and conditionals contained in single sentences differs significantly for an about a fourth of the eligible domains.

A stratified analysis for RQ 4c is reported in Table 4.5a and shows considerable differences in the usage of cue phrases in the domains, but also a degree of overlap: the cue phrase *"if"* is among the five most frequent cue phrases in all domains, closely followed by the cue phrases *"when"* and *"where"*. The stratified frequencies align with the overall distribution reported in Table 4.3 lead to the assumption that the distribution of cue phrases is mostly domain independent. When looking at the most precise cue phrases per domain in Table 4.5b and the least precise cue phrases per domain in Table 4.5c, the cue phrases also reflect the findings from the overall distribution: precise cue phrases like *"if"*, *"when"*, and *"because"* as well as infrequent, but precise causative verbs are equally represented in the domains just as imprecise cue phrases like *"for"* or *"by"*. We conclude that despite slight domain-specific variations, the results for RQ 3c are also domain-independent.[4]

4 More extensive tables reporting on the frequency and Precision of cue phrases in eligible domains are included in our replication package: `https://doi.org/10.5281/zenodo.5596668`.

Table 4.5: Distribution and Precision of Cue Phrases in Eligible Domains.

(a) Relative Frequency of Cue Phrases Within One Domain.

Domain	most frequent	2nd most frequent	3rd most frequent	4th most frequent	5th most frequent
Aerospace	during (8.5%)	when (8.4%)	if (7.5%)	in order to (3.7%)	after (3.0%)
Astronomy	for (10.2%)	during (9.1%)	allow (9.1%)	in (5.7%)	to (5.7%)
Banking	if (11.2%)	to ensure (8.8%)	once (7.2%)	allow (5.6%)	through (4.8%)
Data Analytics	if (10.7%)	when (9.4%)	where (7.9%)	for (5.2%)	during (4.2%)
Health	when (11.5%)	if (11.5%)	during (10.5%)	for (4.6%)	after (3.8%)
Infrastructure	where (16.5%)	if (15.1%)	to ensure (5.6%)	for (5.3%)	then (4.9%)
Smart City	in order to (10.7%)	when (7.4%)	if (7.1%)	therefore (4.9%)	that (4.4%)
Sustainability	therefore (8.6%)	in order to (7.0%)	if (5.9%)	for (5.4%)	where (4.3%)
Telecomm	if (8.7%)	during (8.0%)	in order to (6.0%)	when (5.3%)	in case of (4.7%)

(b) Most Precise Cue Phrases of Each Eligible Domain.

Domain	most precise	2nd most precise	3rd most precise	4th most precise
Aerospace	imposes (100.0%)	as far as (100.0%)	result from (100.0%)	in this case (100.0%)
Agriculture	since (100.0%)	whose (100.0%)	before (100.0%)	when (100.0%)
Astronomy	during (100.0%)	in the event of (100.0%)	so that (100.0%)	attain (100.0%)
Automotive	when (100.0%)	in order to (100.0%)	therefore (100.0%)	whose (100.0%)
Banking	in order to (100.0%)	reduce (100.0%)	will be required (100.0%)	since (100.0%)
Data Analytics	as long as (100.0%)	increases (100.0%)	unless (100.0%)	so that (100.0%)
Digital Library	only for (100.0%)	cause (100.0%)	because (100.0%)	unless (100.0%)
Digital Society	in order to (100.0%)	if (100.0%)	due to (100.0%)	allows (100.0%)
Health	allows (100.0%)	so that (100.0%)	in the event of (100.0%)	as a result (100.0%)
Infrastructure	lead to (100.0%)	prevent (100.0%)	whose (100.0%)	until (100.0%)
Physics	is needed (100.0%)	after (100.0%)	therefore (100.0%)	when (100.0%)
Smart City	result in (100.0%)	to measure (100.0%)	increase (100.0%)	thereby (100.0%)
Sustainability	will require (100.0%)	enables (100.0%)	ensures (100.0%)	because (100.0%)
Telecomm	allows (100.0%)	enables (100.0%)	during (100.0%)	lead to (100.0%)

(c) Least Precise Cue Phrases of Each Eligible Domain.

Domain	least precise	2nd least precise	3rd least precise	4th least precise
Aerospace	according to (14.3%)	to get (20.0%)	to mitigate (20.0%)	based on (24.1%)
Agriculture	on (26.1%)	that (27.1%)	at (28.2%)	allows (33.3%)
Astronomy	from (30.0%)	by (31.0%)	within (40.0%)	where (50.0%)
Automotive	which (10.0%)	for (20.0%)	on (24.2%)	that (31.8%)
Banking	which (21.4%)	to provide (25.0%)	create (28.6%)	by (28.8%)
Data Analytics	to meet (16.7%)	imply (20.0%)	following (20.4%)	but (20.5%)
Digital Library	allow (11.1%)	for (27.6%)	that (30.3%)	with (31.8%)
Digital Society	for (45.8%)	for this reason (50.0%)	where (62.5%)	allow (100.0%)
Health	in this (16.7%)	around (20.0%)	based on (22.2%)	through (25.0%)
Infrastructure	following (18.2%)	to provide (25.0%)	on (41.3%)	in (42.8%)
Physics	while (33.3%)	from (34.8%)	in this (42.9%)	for (43.6%)
Smart City	within (11.1%)	to perform (15.4%)	who (15.4%)	where (15.8%)
Sustainability	within (3.8%)	with (13.7%)	to provide (14.3%)	while (14.3%)
Telecomm	to provide (16.7%)	which (19.4%)	so that (20.0%)	given (20.0%)

4.4 Threats to Validity

Internal Validity A threat to the internal validity is the annotation process itself as any annotation task is subjective to a certain degree. This is especially relevant for more ambiguous categories like Explicit, as *implicit* conditionals are difficult to determine. Two mitigation strategies were performed to minimize the bias of the annotators: First, we conducted a workshop prior to the annotation process to ensure a common understanding of conditionals. Second, we assessed the inter-rater agreement by using multiple metrics (Cohen's Kappa, Agreement Score, and Gwet's AC1). However, it has to be noted that all categories except Conditional Present are dependent on a sentence's classification regarding that category, which may imply a confounding factor for the inter-rater agreement on the other categories. This manifests in the calculation of the inter-rater agreement, where all categories except Conditional Present are calculated based on the 499 identified conditionals. We argue, however, that the other categories are irrelevant for sentences that do not contain a conditional as they only refer to the conditional statement contained by a sentence. Hence, this confounding factor is deemed minimal. Apart from that, the inter-rater agreement is not domain-specific, which implies that it is not possible to identify whether certain domains caused more disagreement among the raters. We deem the general inter-rater agreement reported in Table 4.2 sufficient but recommend considering this aspect for replications and future studies intensifying the domain-dependent aspect of conditionals.

Furthermore, restricting the manual detection of conditional statements to a span of a maximum of two sentences poses also a threat to internal validity, as the potential existence of conditionals that are spread across more than two sentences can neither be confirmed nor denied based on our investigation. We see this threat to be minimal as the relationship between *one-sentence* conditionals and *two-sentence* conditionals allows for the assumption, that the further elements of a conditional are spread apart, the more unlikely the existence of such a conditional is. Extrapolating from the low number of sentences categorized as *two-sentence* conditionals gives us reason to assume that disregarding conditionals spread across three sentences or more is negligible for this initial case study.

External Validity To achieve reasonable generalizability, we selected requirements documents from different domains and years. As Figure 4.1 shows, our data set covers a variety of domains, but the distribution of the sentences is imbalanced. The domains *Aerospace*, *Data Analytics*, and *Smart City* account for a large share in the data set (9,724 sentences), while the other 15 domains are rather underrepresented. We mitigate this threat to validity by including a domain-specific investigation reported in the scope of RQ 6, which confirms that the occurrence of conditionals is to a large degree domain-independent. Future studies should however expand to more documents emerging from underrepresented domains to allow a more general reflection upon different aspects of conditionals in requirements documents.

4.5 Concluding Discussion

Based on the results of our case study, we draw the following conclusions: Conditionals are prevalent in requirements artifacts and therefore matter in requirements engineering,

which motivates the necessity of an effective and reliable approach for the automatic extraction of conditionals in requirements. The complexity of conditional statements is confined since they usually consist of a single *antecedent* and *consequent* relationship in all observed, eligible domains. However, for an approach that aims to extract conditionals to be applicable in practice, it needs to comprehend also more complex relations containing at least two to three and at best an arbitrary number of *antecedents* and *consequents*. Understanding conjunctions, disjunctions, and negations is consequently imperative to fully capture the relationships between *antecedents* and *consequents* and ensure the applicability of a detection and extraction approach.

Two-sentence conditionals and *event chains* occur only rarely. Thus, both aspects can initially be neglected in the development of the approaches and preserve coverage of more than 92 % of the analyzed sentences. The dominance of *explicit* over *implicit* conditionals in the observed sentences simplifies the detection and extraction of conditionals. The information about both *antecedent* and *consequent* is embedded directly in the sentences so that an approach requires little or no *implicit* knowledge. The analysis of the precision values reveals that most of the used cue phrases are ambiguous. Consequently, automatic extraction methods require a deep understanding of language as the presence of certain cue phrases is insufficient as an indicator for conditionals. Instead, a combination of the syntax and semantics of the sentence has to be considered to reliably detect conditional statements.

Chapter 5

Empirical Study on Logical Interpretation of Conditionals in Requirements Artifacts

Common Thread Similar to the previous chapter, this chapter also addresses the first problem that is tackled in this thesis: *"the missing understanding of the notion of conditionals in RE artifacts"* (see Section 1.3). To this end, we conduct a survey with 104 RE practitioners to understand how specific conditionals are interpreted by readers who work with requirements. Specifically, we ask how they interpret 12 different conditional clauses and map their interpretations to logical formulas written in *Propositional (Temporal) Logic*. This allows us to discover which formalization most closely resembles the interpretation of practitioners and thus should be used as a basis for our conditional extraction approach. Moreover, we study different factors (e.g., domain context of the requirement) that might influence the logical interpretation of conditional clauses in requirements.

Contribution The conditionals in our tested requirements were interpreted ambiguously. We found that practitioners disagree on whether an *antecedent* is only *sufficient* or also *necessary* for the *consequent*. We observed a statistically significant relationship between the interpretation and certain context factors of practitioners (e.g., experience in RE, the way how a practitioner interacts with requirements, and the presence of domain knowledge). Interestingly, domain knowledge does not promote a consistent interpretation of conditionals. We found that the choice of certain cue phrases has an impact on the degree of ambiguity (e.g., *"while"* was less ambiguous than *"if"* or *"when"* w.r.t. temporal relationship). In summary, our study reveals that conditionals in requirements are a source of ambiguity and there is not just one way to interpret them formally. Hence, we require two variants of conditional extraction to ensure that the automatically derived test cases correspond to the different logical interpretations: The first variant interprets conditionals as implications and generates only the positive test cases. The second variant interprets the conditionals as equivalences and generates both the positive and negative test cases. The users of our automated conditional extractor should then decide for themselves which test cases conform to their logical interpretations.

Related Publications This chapter is taken, directly or with minor modifications, from a previous publication [5].

5.1 Research Objective

The interpretation of the semantics of conditionals affects all activities carried out on the basis of documented requirements such as manual reviews, implementation, or test case generations. Even more, a correct interpretation is absolutely essential for all automatic analyses of requirements that consider the semantics of sentences; for instance, automatic quality analysis like smell detection [35], test case derivation [1, 284], and dependency detection [2]. In consequence, conditionals should always be associated with a formal meaning to automatically process them. However, determining a suitable formal interpretation is challenging because conditional statements in NL tend to be ambiguous. This can be illustrated by the following conditional: *"If the system detects an error (e_1), an error message shall be shown (e_2)".*

Literally, the conditional may be interpreted as a logical implication ($e_1 \Rightarrow e_2$), in which e_1 is a *sufficient* precondition for e_2. However, it is equally reasonable to assume that the error message shall not be shown if the error has not been detected (i.e., e_1 is a *sufficient* and also *necessary* condition for e_2). Furthermore, it is reasonable to assume that e_1 must occur *before* e_2. Both assumptions are not covered by an implication as it neglects temporal ordering. In contrast, the assumptions need to be expressed by temporal logic (e.g., LTL [91]). Existing guidelines for expressing requirements have different ways of interpreting conditionals; for instance, Mavin et al. [152] propose to interpret conditionals as a logical equivalence ($e_1 \Leftrightarrow e_2$) to avoid ambiguity. We argue that the *"correct"* way of interpretation should not just be defined by the authors of a method, but rather from the view of practitioners. This requires an understanding of how these interpret such conditionals. Otherwise, we choose a formalization that does not reflect how practitioners interpret conditionals, rendering downstream activities error-prone. That is, we would likely derive incomplete test cases or interpret dependencies between the requirements incorrectly.

We aim to understand and (logically) formalize the interpretation of conditionals in requirements by RE practitioners in software development projects. To this end, we conducted a survey following the guidelines by Ciolkowski et al. [285]. The expected outcome of our survey is a better understanding of how practitioners logically interpret conditional clauses in requirements. Further, we aim to determine which of the elements in our formalization matrix (introduced in Section 2.1) match their logical interpretations (see Figure 5.1). We derived three research questions (RQ) from our survey goal.

- **RQ 1:** How do practitioners logically interpret conditional clauses in requirements?
- **RQ 2:** Which factors influence the logical interpretation of conditional clauses in requirements?
- **RQ 3:** Which (if any) cue phrases promote (un)ambiguous interpretation?

RQ 1 investigates how conditionals are interpreted by practitioners and how their interpretations should be formalized accordingly. RQ 2 studies whether the logical interpretation of practitioners depends on certain factors. We focus on: 1) the role of the participant (e.g., writing requirements vs. reading and implementing requirements) and 2) the domain context of the requirement (i.e., does the requirement describe system behavior from a domain that is familiar to the participant or does the requirement originate

Table 5.1: Overview of Study Objects. Cue Phrases Are Highlighted in **Bold**.

DS 1	[S1]	**After** unlocking a door with the inside door handle, the Automatic Door opens until it reaches maximum opening position, or it detects an obstacle.
	[S2]	Closing is done in a fast manner **if** no passenger is inside the car.
	[S3]	The door handle is extended **when** a car key is detected in proximity of the car and a hand is near the position of the door handle.
	[S4]	**While** the button is in manual movement down position, the window is moved down.
DS 2	[S5]	**When** a conflict is detected and an alert is issued, additional resolution information is needed. (retrieved from REQ-DOC-22)
	[S6]	The Manual-Electric-Mode-Controller, **while** engaged, provides rudimentary transformation of the AFD control inputs. (REQ-DOC-26)
	[S7]	Level 1.5 data are disseminated to users **after** being rectified to 0° longitude. (REQ-DOC-27)
	[S8]	**If** the sampling operation was not successful, the spacecraft can undertake 2 more attempts. (REQ-DOC-30)
DS 3	[S9]	**After** event 1, event 2 or event 3 occur.
	[S10]	Event 1 occurs **if** event 2 is present.
	[S11]	Event 1 occurs **when** event 2 is present.
	[S12]	**While** event 1 is present, event 2 occurs.

from an unknown domain?). RQ 3 aims at the formulation of conditionals: Conditional clauses can be expressed by using different cue phrases (e.g., *"if"*, *"when"*). We hypothesize that cue phrases impact the logical interpretation of practitioners. With RQ 3, we want to identify cue phrases for which the interpretations are almost consistent, and cue phrases that are ambiguous. This insight enables us to derive best practices on writing conditionals in requirements specifications.

5.2 Survey Design

Target Population and Sampling The selection of survey participants was driven by a purposeful sampling strategy [286] along with the following criteria: a) they elicit, maintain, implement, or verify requirements, and b) they work in industry and not exclusively in academia. Each author prepared a list of potential participants using their personal or second-degree contacts (convenience sampling [287]). From this list, the research team jointly selected suitable participants based on their adequacy for the study. To increase the sample size further, we asked each participant for other relevant contacts after the survey (snowball sampling). Our survey was started by 168 participants of which 104 completed the survey. All figures in this chapter refer to the 104 participants that completed the survey. The majority of participants were non-native English speakers (94.2 %). We received responses mainly from practitioners working in Germany (94.2 %). The remaining 5.8 % of survey completions originate from Croatia, Austria, Japan, Switzerland, the United States, and China. The experience of the participants in

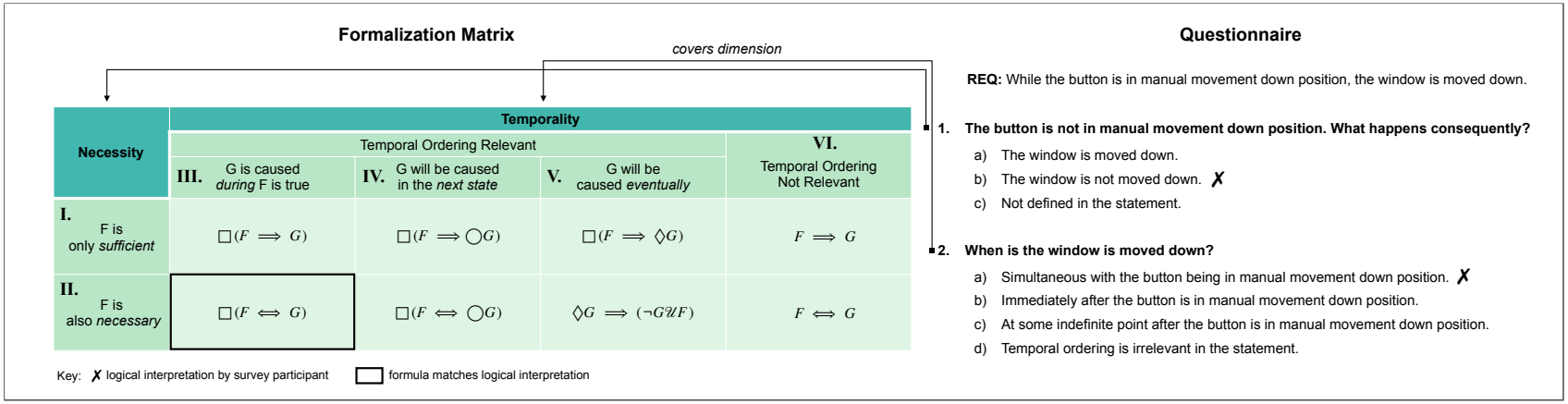


Figure 5.1: Mapping Between Questionnaire (Right) and Formalization Matrix (Left).

RE and RE-related fields is equally distributed: 18.2 % have less than 1-year experience, 26 % between 1 and 3 years, 25 % between 4 and 10 years, and 30.8 % more than 10 years. The participants work for companies operating in 22 different domains. The majority of our participants are employed in the automotive (21 %) and insurance/reinsurance (10.1 %) industry. Over the past three years, our participants have worked in 18 different roles. Most frequently, they had roles as developers, project managers, requirements engineers/business analysts, or testers. 77.9 % of the survey participants elicit requirements as part of their job. 59.6 % verify whether requirements are met by a system. 46.2 % read requirements and implement them. 45.2 % maintain the quality of requirements.

Study Objects To conduct the survey and answer the RQs, we used three data sets (DS), each from a different domain (see Table 5.1). DS 1 contains conditionals from a requirements document describing the behavior of an automatic door in the automotive domain. We argue that all participants have an understanding of how an automatic car door is expected to work so that all participants should have the required domain knowledge. DS 2 contains conditionals from *Aerospace* systems. We hypothesize that no or only a few participants have deeper knowledge in this domain, making DS 2 well suited for an analysis of the impact of domain knowledge on logical interpretations. DS 3 contains abstract conditionals (e.g., *"If event A and event B, then event C"*). Thus, they are free from any domain-induced interpretation bias. To address RQ 3, we focused on four cue phrases in the conditionals: *"if"*, *"while"*, *"after"*, and *"when"*. To avoid researcher bias, we created the data sets by extracting conditionals randomly from existing requirement documents used in practice. The conditionals in DS 1 are taken from a requirements document written by *Mercedes-Benz Passenger Car Development*.[1] The conditionals contained in DS 2 originate from three requirements documents published by NASA and one by ESA.[2] The conditionals in DS 3 are syntactically identical to the conditionals in DS 1, except that we replace the names of the events with abstract names. DS 1–3 contain four conditionals each, resulting in a total of 12 study objects (see Table 5.1). Each cue phrase occurs exactly once in each DS.

Questionnaire Design We chose an online questionnaire as our data collection instrument to gather quantitative data on our research questions. For the design, we followed the guidelines of Dillman et al. [289] to reduce common mistakes when setting up a questionnaire. Since our research goal is of descriptive nature, most questions are closed-ended. We designed three types of questions (Q) addressing the two dimensions and prepared a distinct set of responses (R), among which the participants can choose. Each of these responses can be mapped to a characteristic in the formalization matrix and thus allows us to determine which characteristic the practitioners interpret as being reflected by a conditional (see Figure 5.1). We build the questionnaire for each study object (e.g., *"If F then G"*) according to a pre-defined template (see Figure 5.2). The template is structured as follows: The first question (**Q 1**) investigates the dimension of *Necessity*: if event G cannot occur without event F, then F is not only *sufficient*, but also neces-

1 Thanks to Frank Houdek for sharing the document at NLP4RE'19 [288]: `https://nlp4re.github.io/2019/uploads/demo-spec-automatic-door.pdf`

2 We retrieved these documents from our gold standard corpus of requirements presented in Chapter 4. We are referring to the documents: REQ-DOC-22, REQ-DOC-26, REQ-DOC-27, and REQ-DOC-30.

Q1: F does not occur. What happens consequently?
- **R1.1**: G occurs nevertheless. (sanity check)
- **R1.2**: G does not occur. (→ II)
- **R1.3**: Not defined in the statement. (→ I)

Q2: When does G occur?
- **R2.1**: Simultaneously with F. (→ III)
- **R2.2**: Immediately after F. (→ IV)
- **R2.3**: At some indefinite point after F. (→ V)
- **R2.3**: Temporal ordering is irrelevant in the statement. (→ VI)

Figure 5.2: Questionnaire Template. The Note After Each Answer Option (e.g., → IV) Indicates the Matching Characteristic in the Formalization Matrix (see Figure 5.1). If a Participant Selects R1.2, for Example, She Implicitly Interprets F as *necessary* for G. The Notes Were Not Included in the Questionnaire.

sary for G. We add *"nevertheless"* as a third response option (see R1.1 in Figure 5.2) to perform a sanity check on the answers of the respondents. We argue that interpreting that the *consequent* should occur although the *antecedent* does not occur indicates that the sentence has not been read carefully. The second question (**Q 2**) covers the temporal ordering of the events. In this context, we explicitly ask for the three temporal relations *eventually*, *always* and *next state* described in Section 2.1. Should a participant perceive temporal ordering as irrelevant for the interpretation of a certain conditional, we can conclude that PL is sufficient for its formalization. We ask **Q 1**–**2** for each of the 12 study objects, resulting in a total of 24 questions. To get an overview of the background of our respondents, we also integrated five demographic questions. In total, our final questionnaire consists of 29 questions and can be also found in our replication package.[3]

5.3 Survey Implementation and Execution

We prepared an invitation letter to ask potential participants if they would like to join our survey. We incorporated all of our 29 questions into the survey tool *Unipark* [290]. To avoid bias in the survey data, we allow *Unipark* to randomize the order of the non-demographic questions. We opened the survey on Feb 01, 2021 and closed it after 15 days. We approached all eligible contacts from our prepared list either by e-mail or via *Linkedin* direct message. We also distributed the questionnaire via a mailing list in the RE focus group of the *German Informatics Society* (GI). As the traffic on our survey website decreased during the first week, we contacted all candidates again on Feb 08 in 2021.

5.4 Survey Analysis

To answer the proposed research questions (see Section 5.1), we analyzed the gathered quantitative data as follows.

3 Our replication package contains (1) our final questionnaire, (2) the survey protocol, and (3) the survey responses. It can be found at `https://doi.org/10.5281/zenodo.5070235`.

Analysis for RQ 1 We use heatmaps to visualize how the respondents logically interpret the individual study objects (see Figure 5.3). Each cell in the heatmaps corresponds to a single 2-tuple. Based on the heatmaps, we analyze the logical interpretations of the participants and decide which formalization should be chosen for each study subject according to the most frequent 2-tuple.

Analysis for RQ 2 We focus on three factors (f_n) and investigate their impact on the logical interpretations of practitioners: (1) the experience in RE (f_1: **Experience**), (2) how the practitioners interact with requirements (elicit, maintain, verify,...) in their job (f_2: **Interaction**), and (3) the domain context of the conditional (f_3: **Domain**). To answer RQ 2, we examine the impact of f_1–f_3 on the dimensions described in Section 2.1. In our survey, we collected the dimensions for each sentence individually, resulting in 12 categorical variables per dimension (e.g., $\textbf{nec}_{\textbf{s1}}$, $\textbf{nec}_{\textbf{s2}}$, ... $\textbf{nec}_{\textbf{s12}}$). To get an insight across all sentences, we aggregated all 12 categorical variables per dimension to one variable (resulting in **Necessity**, and **Temporality**). This allows us to analyze, for example, whether the experience of the respondents has an impact on understanding an *antecedent* only as *sufficient* for a *consequent* or as both *sufficient* and *necessary*. In other words, does the perception of **Necessity** depend on **Experience**?

As shown in Table 5.2, all five variables (3x factors and 2x dimensions) are categorical with a maximum of four levels. The majority is nominally scaled, while **Experience** follows an ordinal scale. The variable **Domain** was not gathered directly from the responses, but implicitly from our selection of the data sets. We thus add **Domain** as a variable to our data set, using a categorical scale with three levels: domain knowledge is present (in the case of DS 1), domain knowledge is not present (DS 2), and domain knowledge is not necessary (DS 3). By introducing this new variable, we are able to investigate the relationship between domain knowledge and logical interpretations. We use the *chi-squared test of independence* (χ^2) to analyze the relationship between all variables. We run the test by using SPSS and test the following hypotheses (H_n):

Algorithm 1: Hypothesis testing

```
for f_n ∈ {Experience, Interaction, Domain} do
  for v ∈ {Necessity, Temporality} do
    H_0: The interpretation of v is independent of f_n.
    H_1: The interpretation of v depends on f_n.
  end for
end for
```

We set the p-value at 0.05 as the threshold to reject the null hypothesis. To test our hypotheses, we need to calculate the contingency tables for each combination of f_n and dimension. The total number of survey answers per dimension is 1,248 (104 survey completions * 12 annotated sentences). Since we allow the respondents to specify multiple ways to interact with requirements (e.g., to both elicit and implement requirements), our survey data contains a multiple dichotomy set for **Interaction**. In other words, we created a separate variable for each of the selectable interaction ways (four in total for verify, maintain, elicit and implement). Each variable has two possible values (0 or 1), which indicate whether or not the response was selected by the participant. Therefore, we define a multiple response set in SPSS to create the contingency table for **Interaction**. The χ^2

Table 5.2: Overview of Analyzed Variables.

Name	Levels	Type	Scale
Experience	• less than 1 year • 1–3 years • 4–10 years • more than 10 years	categorical (single select)	ordinal
Interaction	• elicit • maintain • verify • implement	categorical (multiple select)	nominal
Domain	• domain knowledge present • domain knowledge not present • domain knowledge not necessary	categorical (single select)	nominal
Necessity	• nevertheless • only sufficient • also necessary	categorical (single select)	nominal
Temporality	• during • next state • eventually • temporal ordering not relevant	categorical (single select)	nominal

test allows us to determine if there is enough evidence to conclude an association between two categorical variables. However, it does not indicate the strength of the relationship. To measure the association between our variables, we use Cramer's Phi ϕ [291] in the case of two nominally scaled variables and Freeman's theta Θ [292] in case of one ordinally scaled and one nominally scaled variable. We calculate ϕ by using SPSS and Θ by using the R implementation *"freemanTheta"*. We interpret Θ according to the taxonomy of Vargha and Delaney [293]. For the interpretation of ϕ, we use the taxonomy of Cohen [291].

Analysis for RQ 3 A conventional way to measure ambiguity is by calculating the inter-rater agreement (e.g., Fleiss Kappa [294]). However, inter-rater agreement measures must be used carefully, as they have a number of well-known shortcomings [276]. For example, the magnitude of the agreement values is not meaningful if there is a large gap between the number of annotated units and the number of involved raters. In our case, we examine only three units per cue phrase (i.e., *"if"* is only included in S2, S8, and S10), each of which was annotated by 104 raters. This discrepancy between the number of units and raters leads to a very small magnitude of the agreement values and distorts the impression of agreement. For example, if we calculate Fleiss Kappa regarding the dimension *Temporality* of sentences that contain the cue phrase *"while"*, we obtain a value of 0.053. According to the taxonomy Landis and Koch [280], this would imply only a slight agreement between the raters. However, there is a substantial agreement among the raters that *"while"* indicates a simultaneous relationship. This can be demonstrated by the distribution of survey answers across the different *Temporality* levels (see Figure 5.4).

Thus, instead of reporting less meaningful inter-rater agreement measures, we provide histograms visualizing the distribution of ratings on the three investigated dimensions. We create the histograms for each set of study objects containing the same cue phrase. This allows us to analyze which cue phrase produced the highest/lowest agreement for a certain dimension.

5.5 Survey Results

We report on the results of our study structured by our research questions (**RQ 1 - 3**).

RQ 1: How Do Practitioners Logically Interpret Conditional Clauses in Requirements?

In order to answer RQ 1, we first look at the total number of answers for each dimension across all data sets. Secondly, we analyze the distribution of ratings based on our constructed heatmaps (see Figure 5.3).

Necessity Our participants did not have a clear tendency whether an *antecedent* is only *sufficient* or also *necessary* for the *consequent*. Among the total of 1,248 answers, 2.1 % correspond to the level *"nevertheless"*, 46.9 % to *"also necessary"*, and 51 % for *"only sufficient"*. That means that more than half of the respondents stated that the conditional does not cover how the system is expected to work if the *antecedent* does not occur (i.e, the negative case is not specified).

Temporality We found that time plays a major role in the interpretation of conditionals in requirements. Among the 1,248 answers, only 13 % were *"temporal ordering is irrelevant"* for the interpretation. This indicates that conditionals in requirements require temporal logics for a suitable formalization. For some study objects, the exact temporal relationship between *antecedent* and *consequent* was ambiguous. For S3, 34 participants selected *"during"*, 43 *"next state"*, and 19 *"eventually"*. Similarly, we observed divergent temporal interpretations for S2, S5, S7, S10, S11, and S12. In contrast, the respondents widely agreed on the temporal relationship of S1 (67 survey answers for *"next state"*), S4 (84 survey answers for *"during"*), S6 (73 survey answers for *"during"*), S8 (67 survey answers for *"eventually"*) and S9 (83 survey answers for *"eventually"*). Across all study objects, 29.8 % of survey answers were given for the level *"during"*, 20.1 % for *"next state"* and 37.1 % for *"eventually"*.

Agreement Our heatmaps illustrate that there are only a few study objects for which more than half of the respondents agreed on a 2-tuple (see Figure 5.3). This trend is evident across all data sets. The presence or absence of domain knowledge does not seem to have an impact on a consistent interpretation. The greatest agreement was achieved in the case of S1 (48 survey answers for ⟨necessary, next state⟩), S6 (49 survey answers for ⟨necessary, during⟩), S8 (53 survey answers for ⟨sufficient, eventually⟩) and S9 (56 survey answers for ⟨sufficient, eventually⟩). However, for the majority of study objects, there was no clear agreement on a specific 2-tuple. For S5, two 2-tuples were selected equally often, and for S10, the two most frequent 2-tuples differed by only two survey answers.

Generally Valid Formalization? Mapping the most frequent 2-tuples in the heatmaps to our constructed formalization matrix reveals that all study objects can not be formalized in the same way. The most frequent 2-tuples for each study object yield the following six patterns:

- **Pattern 1:** ⟨necessary, next state⟩: S1, S3

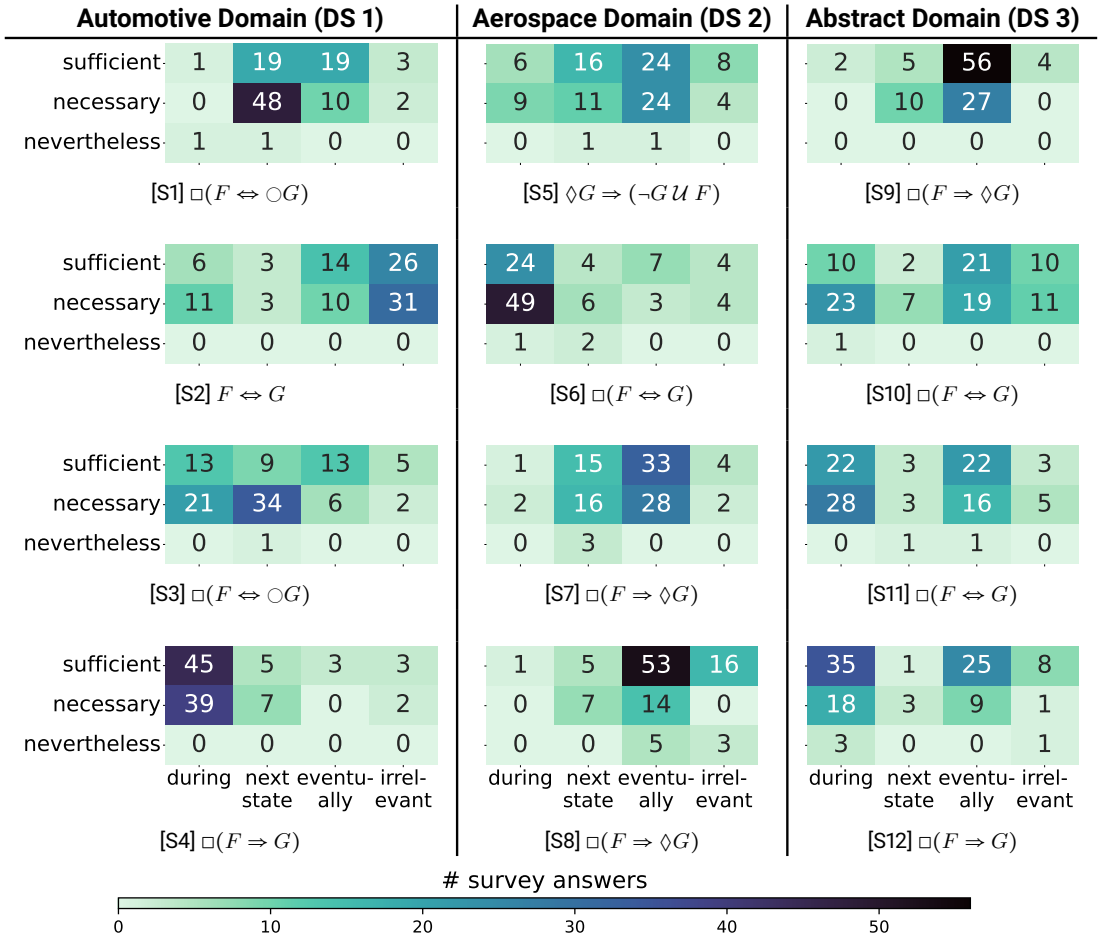


Figure 5.3: Heatmaps Visualizing the Interpretations of the Participants per Study Object [S_n].

- **Pattern 2:** ⟨necessary, irrelevant⟩: S2
- **Pattern 3:** ⟨necessary, during⟩: S6, S10, S11
- **Pattern 4:** ⟨necessary, eventually⟩: (S5)
- **Pattern 5:** ⟨sufficient, eventually⟩: (S5), S7, S8, S9
- **Pattern 6:** ⟨sufficient, during⟩: S4, S12

One sees immediately that it is not possible to derive a formalization for conditionals in general. Especially the temporal interpretations differed between the conditionals and the used cue phrases (see Figure 5.4). However, it can be concluded that, except for S2, the interpretations of all study objects can be represented by LTL.

RQ 2: Which Factors Influence the Logical Interpretation of Conditional Clauses in Requirements?

This section reports the results of our chi-square tests (see Table 5.3). In our contingency tables, no more than 20 % of the expected counts are < 5. Hence, we satisfy the assumption

Table 5.3: Results of Chi-Square Test of Independence. Statistically Significant Relationships Between Factors and Interpretation Are Marked in **Bold**.

	Test Statistics			Measures	
Tested Relationship	χ^2	**df**	**p-value**	ϕ	Θ
Experience and Necessity	2.384	6	0.881	-	-
Experience and Temporality	31.523	9	0.001	-	0.089
Interaction and Necessity	11.005	8	2.201	-	-
Interaction and Temporality	36.991	12	< 0.001	0.510	-
Domain and Necessity	22.310	4	< 0.001	0.134	-
Domain and Temporality	138.128	6	< 0.001	0.333	-

of enough observations per category for the chi-square test [295]. In the following, we explain the relationships where the chi-square test indicated a dependency between the logical interpretation and a factor.

The Logical Interpretation Regarding Temporality Depends on RE Experience In the group with less than 1 year of experience, there is a tendency to perceive the temporal relationship between the events as *"during"* (36.4 %). In the group of participants with 4–10 years of experience, most of the respondents rated the temporal relationship as *"eventually"* (41.3 %). The χ^2 test reveals that the distribution of ratings differs between the experience levels. The calculated Θ value indicates that the strength of the relationship is low.

The Logical Interpretation Regarding Temporality Is Dependent on How a Practitioner Interacts With Requirements Our contingency table reveals that the distribution of ratings differs between the interaction levels. Practitioners who implement requirements fluctuate mainly between *"during"* and *"eventually"*, while they rarely selected the other two *Temporality* levels. A different pattern emerges for practitioners who maintain and verify requirements. Across all study objects, they choose the levels *"during"*, *"next state"* and *"eventually"* equally often. A χ^2 test indicates a dependency between both variables. The calculated ϕ value indicates that the strength of the relationship is high.

The Logical Interpretation Regarding Necessity Is Dependent on Domain Knowledge The disagreement about whether an *antecedent* is only *sufficient* or also *necessary* holds regardless of domain knowledge. However, the trend differs between the data sets with respect to the *Necessity* levels. In the case of DS 1 (domain knowledge assumed), more answers were given for *"also necessary"* (54.3 %) than for *"only sufficient"* (45 %). In contrast, more ratings were given for *"only sufficient"* in the case of DS 2 (53.1 %) and DS 3 (55 %). The slight difference in the distribution of the ratings regarding *Necessity* is supported by the χ^2 test. However, the strength of the relationships is low.

The Logical Interpretation Regarding Temporality Is Dependent on Domain Knowledge Our contingency table shows that the distribution of ratings regarding *Temporality* differs between the data sets. In the case of DS 1, ratings were mainly given for *"during"* (32.9 %)

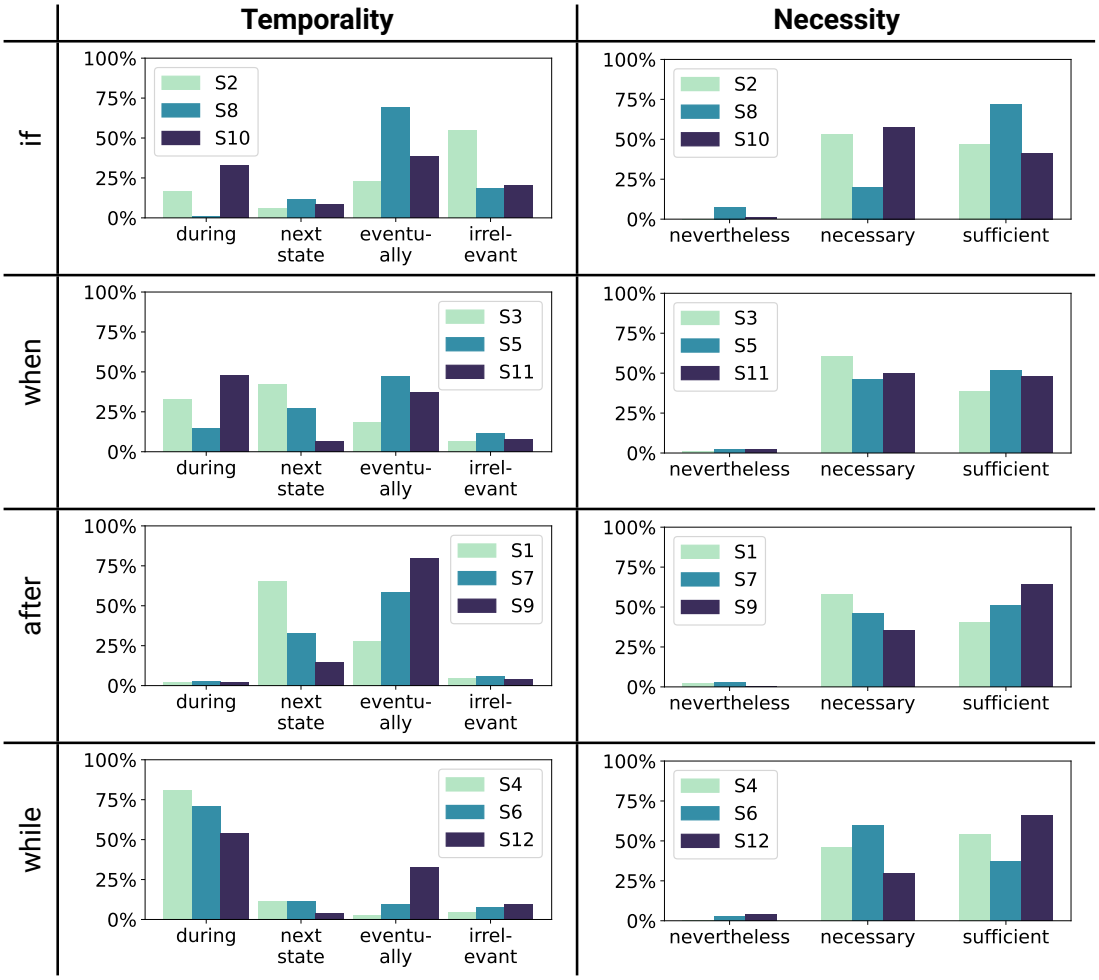


Figure 5.4: Distribution of Survey Answers on the Different Variable Levels for Each Set of Study Objects With the Same Cue Phrase (e.g., S2, S8 and S10 Include *"If"*).

and *"next state"* (31.3 %). In the case of the unknown domain (DS 2), ratings were mainly assigned to *"eventually"* (46.2 %), while only 20.7 % were given to *"next state"* and 22.4 % to *"during"*. In DS 3, where no domain knowledge is necessary for the understanding of the conditionals, most ratings were given to *"during"* (34.1 %) and *"eventually"* (47.1 %). A χ^2 test shows that there is a statistically significant dependency between both variables. According to the calculated ϕ value, the strength of the relationship is medium.

RQ 3: Which (If Any) Cue Phrases Promote (Un)Ambiguous Interpretation?

The histograms in Figure 5.4 show that the logical interpretation regarding *Temporality* depends on the cue phrase used to express a conditional. For study objects containing *"while"* (S4, S6 and S12), the respondents largely agreed that the *consequent* occurs simultaneously with the *antecedent*. In contrast, almost no respondent associated simultaneous events in the study objects with the cue phrase *"after"*. Instead, the respondents vacillated between the temporal levels *"next state"* and *"eventually"*. The largest disagreement, though, was found in the interpretations of the conditionals *"if"* or *"when"*. Es-

pecially in the case of *"when"*, there was no clear agreement across S3, S5 and S11 on whether *antecedent* and *consequent* are in a *"during"*, *"next state"* or *"eventually"* temporal relationship. Regarding *Necessity*, we observe that the practitioners, irrespective of the used cue phrase, disagree whether the *antecedent* is only *sufficient* or also *necessary* for the *consequent*. We found one outlier in our histograms (S8), where an 80 % agreement for the level *"sufficient"* could be achieved. For the remaining study objects, however, there is a balanced number of survey answers for both levels.

5.6 Threats to Validity

Internal Validity The respondents may have misunderstood the questions resulting in poor quality or invalid answers. To minimize this threat, we followed the guidelines by Dillmann [289] in the creation of the questionnaire. In addition, we conducted a pilot phase to validate the questionnaire internally through discussions in the research team and externally through pilot survey runs. Selection bias is another threat. Although we have started with personal contacts to find participants, the sampling process has been extended by indirect contacts. As a result, selection bias has been reduced. Another possible threat is the selection of dimensions by which we formalize conditionals. The two dimensions, *Temporality* and *Necessity*, have been selected after extensive literature research and discussion among the authors. However, the completeness of dimensions can neither be proven nor rebutted. One threat that we were unable to control was the distribution of native speakers. Although one could argue that non-native speakers reading and writing English requirements are the standard case for most projects, and therefore, their interpretation is meaningful nevertheless, future research should validate the findings also with a dedicated group of native speakers.

Another threat arises from our assumption that each participant has the necessary domain knowledge in the case of DS 1 but lacks it in the case of DS 2. To mitigate this threat, we analyzed the feedback received during our pilot study. In the case of DS 1, almost no questions were raised, whereas in the case of DS 2, many pilot users lacked knowledge about the described system behavior. This indicates that the respondents may have the necessary domain knowledge to interpret the conditionals described in DS 1. Furthermore, the conditionals in DS 1 are derived from the data set used in the tool competition at the NLP4RE workshop, which is claimed to be interpretable without specific domain knowledge.

External Validity As in every survey, the limited sample size and sampling strategy do not provide the statistical basis to generalize the results of the study. However, we tried to involve RE practitioners working in different roles at companies from different domains to obtain a comprehensive picture of how conditionals are logically interpreted. We argue that our survey sample of 104 RE practitioners, who work in 22 different domains and of which a third have more than 10 years of experience in RE is sufficient for a first insight into the logical interpretation of conditionals.

Construct Validity The questionnaire might not sufficiently cover our research questions limiting the availability of data that provides suitable answers to the research questions. To minimize this threat, we constructed a formalization matrix and designed our

questionnaire according to the dimensions of the matrix to establish a distinct mapping between interpretation and suitable formalization.

5.7 Concluding Discussion

As outlined in Chapter 4, conditionals are common to specify desired system behavior in RE artifacts. In this chapter, we show that conditionals are interpreted ambiguously by RE practitioners. In particular, there is disagreement (1) about whether an *antecedent* is only *sufficient* or also *necessary* for a *consequent*, and (2) about the temporal occurrence of *antecedent* and *consequent* when different cue phrases (such as *"when"* or *"if"*) are used. Thus, a generic formalization of conditionals will inevitably fail at least some practitioner's interpretation. We see two immediate implications in practice:

Implications for Automatic Methods Especially (if not limited) for automated test case generation, it is vital to understand which behavior is desired if the *antecedent* does not occur. The evidence presented in this paper refutes the prevailing assumption (cf. [152, 2]) that *antecedents* can always be treated as *necessary* conditions. Hence, we propose that future methods should display the automatically generated positive and negative test cases to practitioners and explicitly verify: *"Is the negative case of your conditional also valid?"*. This will foster the discussion within project teams about the expected system behavior and enables to resolve misunderstandings at an early stage. We consider this finding when developing our approach for the automatic generation of acceptance tests and integrated it into the *User Interface* of our tool (see Section 8.2).

Implications for Requirements Authors It should be incorporated into RE writing guidelines that it does matter which cue phrase is used for the formulation of a conditional. *"While"* is interpreted consistently, but *"if"* and *"when"* cause misunderstandings about the temporal interpretation of *antecedent* and *consequent*. This poses a problem especially in the implementation of requirements and eventually leads to discrepancies between actual and expected system behavior. Project teams should therefore agree early on how they want to interpret the different cue phrases to avoid ambiguities. Additionally, our findings provide empirical evidence for the claim by Berry et al. [296] and Rosadini et al. [297] that requirements authors should always specify the negative case (e.g., by using an else-statement) to prevent confusion about the necessity of *antecedents*.

Part II

Extracting Conditionals From Requirements Artifacts

Common Thread As outlined in Section 1.3, we require an approach capable of extracting conditionals in fine-grained form to make them useful for test case derivation and dependency detection between requirements. However, existing approaches are not suitable for this purpose. In this part of the thesis, we address this problem and present our tool-supported approach named `CiRA` (*Conditionals in Requirements Artifacts*). `CiRA` solves conditional extraction as a two-step problem: It first detects whether requirements contain conditional statements. Second, if they contain conditionals, it interprets and extracts them in fine-grained form. We implement different methods for both steps and compare them with each other in experiments. For the detection part, we compare the performance of rule based approaches, ML-based approaches (e.g., *Random Forest*, *Support Vector Machines*), and TL-based approaches (e.g., BERT). We report on the results of our experiments in Chapter 6. For the extraction part, we implement three approaches and compare their performance with each other: *Dependency Parsing*, RNTN, and TL-based approaches (e.g., RoBERTa, DistilBERT). The results of our comparison can be found in Chapter 7. Based on the results of our experiments, we select the best approach for both steps and incorporate them into the `CiRA` pipeline. Chapter 8 condenses the functionality of `CiRA` and introduces a tool that allows fellow researchers and practitioners to easily interact with `CiRA`.

Preliminaries To understand this part, the reader needs to possess a solid understanding of NLP methods. Chapter 6 requires knowledge of the core idea behind *Transfer Learning* (see Section 2.3.3). To understand the extraction part, the difference between *Dependency Parsing* and *Constituency Parsing* needs to be understood (see Section 2.3.1). Further, one needs to comprehend the forward and backward propagation of an RNTN to follow our RNTN-based conditional extraction approach (see Section 2.3.2).

Chapter 6

Automatic Detection of Conditionals in Requirements Artifacts

Common Thread This chapter focuses on the first step in the CiRA pipeline: the detection of conditionals in NL requirements. For this purpose, we implement and compare different NLP methods such as rule-based approaches, ML-based approaches (e.g., *Random Forest*, *Support Vector Machines*), and TL-based approaches (e.g., BERT). We select the method with the best performance as our final conditional classifier and incorporate it into the CiRA pipeline.

Contribution We train and evaluate our methods on a real-world data set of 8,430 sentences retrieved from requirements documents. The data set is *balanced* (i.e., the distribution of sentences containing conditionals (4,215) and sentences without conditionals (4,215) is equal). Our study demonstrates that the rule-based approach is not able to distinguish between sentences that contain conditionals (F_1 score: 66 %) and sentences that do not (F_1 score: 64 %). In contrast, we found that syntactically enriched BERT embeddings combined with a softmax classifier outperform other methods in detecting conditional statements (macro-F_1 score: 82 %). However, the application of TL does not result in a large performance boost over conventional ML methods (gain of only 4 % in macro-F_1 compared to the best ML method).

Related Publications This chapter is taken, directly or with minor modifications, from previous publications [4, 7].

6.1 Principal Idea

We understand the detection of conditionals in RE artifacts as a binary classification problem: we are given a certain RE artifact $\mathcal{X}$ and are required to produce a nominal label $y \in \mathcal{Y} = \{$Conditional Present, Conditional Not Present$\}$. Specifically, we need to implement a classifier capable of demarcating RE artifacts that do contain a conditional from RE artifacts that do not. Since requirements are still mostly expressed in *Natural Language* [61, 62], our conditional classifier needs to be able to interpret NL. Thus, we employ different NLP methods to implement our classifier. We train our methods on the data set of 14,983 requirements which we annotated in our first empirical study on the prevalence of conditionals in RE artifacts (see Chapter 4).

6.2 Implementation

This section presents the implementation of our conditional classifier. Specifically, we describe three methods that we use to detect conditional statements in NL sentences: a rule-based, ML-based, and TL-based approach.

Rule-Based Approach Our baseline approach is built on the idea that the presence of certain cue phrases can be used to infer the occurrence of a conditional. We thus follow up on the results of our study on the prevalence of conditionals (see Chapter 4) and use simple regex expressions to implement our baseline approach for conditional detection. We iterate through all sentences in the test set and check if one of the phrases listed in Table 4.3 is included. For the positive case, the sentence is classified as Conditional Present and vice versa.

Machine Learning-Based Approach As a second approach, we investigate the use of *supervised* ML models that learn to predict conditionals based on the labeled data set. Specifically, we employ established binary classification algorithms: *Naive Bayes* (NB), *Support Vector Machines* (SVM), *Random Forest* (RF), *Decision Tree* (DT), *Logistic Regression* (LR), *Ada Boost* (AB) and *K-Nearest Neighbor* (KNN). To determine the best hyperparameters for each binary classifier, we apply *Grid Search*, which fits the model on every possible combination of hyperparameters and selects the most performant. We use two different methods as word embeddings: *Bag-of-Words* (BoW) and TF-IDF. In Table 6.1 we report the classification results of each algorithm as well as the best combination of hyperparameters.

Transfer Learning-Based Approach We make use of the fine-tuning mechanism of BERT and investigate to which extent it can be used for conditional detection in requirement sentences. First, we tokenize each sentence by using the *WordPiece* Tokenizer. As described in Section 2.3.3, BERT requires input sequences with a fixed length (maximum 512 tokens). Therefore, for sentences that are shorter than this fixed length, PAD tokens are inserted to adjust all sentences to the same length. Other tokens, such as the CLS token, are also inserted in order to provide further information about the sentence to the model. CLS is the first token in the sequence and represents the whole sentence (i.e., it is the pooled output of all tokens of a sentence). For our classification task, we mainly

use the CLS token because it stores the information of the whole sentence. We feed the pooled information into a single-layer feedforward neural network that uses a softmax layer, which calculates the probability that a sentence contains a conditional or not:

$$\hat{y} = \text{softmax}(W v_0 + b) \tag{6.1}$$

$\hat{y}$ are the predicted class probabilities for the sentence, W is the weighted matrix, v_0 represents the first token in the sentence (i.e., the CLS token), and b is the bias. We select the class (c) with the highest probability as the final classification result:

$$c = \text{argmax}(\hat{y}) \tag{6.2}$$

We tune BERT in three different ways and investigate their performance:

$\text{BERT}_{\text{Base}}$ In the base variant, the sentences are tokenized as described above and put into the classifier. To choose a suitable fixed length for our input sequences, we analyzed the lengths of the sentences in our data set. Even with a fixed length of 128 tokens, we cover more than 97 % of the sentences. Sentences containing more tokens are shortened accordingly. Since this is only a small amount, only a little information is lost. Thus, we chose a fixed length of 128 tokens instead of the maximum possible 512 tokens to keep the computational requirements of BERT to a minimum.

BERT_{POS} Studies have shown that the performance of NLP models can be improved by providing explicit prior knowledge of syntactic information to the model [298]. Therefore, we enrich the input sequence with syntactic information and feed it into BERT. More specifically, we add the corresponding POS tag to each token by using the *spaCy* NLP library [123]. One way to encode the input sequence with the corresponding POS tags is to concatenate each token embedding with a hot encoded vector representing the POS tag. Since the BERT token embeddings are high dimensional, the impact of a single added feature (i.e., the POS tag) would be low. Contrary, we hypothesize that the syntactic information has a higher impact if we annotate the input sentences directly with the POS tags and then put the annotated sentences into BERT. This way of creating linguistically enriched input sequences has already proven to be promising during the development of the word embeddings published by *Nordic Language Processing Laboratory* (NLPL) [299]. Figure 6.1 shows how we incorporated the POS tags into the input sequence. By extending the input sequence, the fixed length for the BERT model has to be adapted accordingly. After further analysis, a length of 384 tokens proved to be reasonable.

BERT_{DEP} Similar to the previous fine-tuning approach, we follow the idea of enriching the input sequence with linguistic features. Instead of using the POS tags, we use the *Dependency* (DEP) tags (see Figure 6.1) of each token. Thus, we provide knowledge about the grammatical structure of the sentence to the classifier. We hypothesize that this knowledge has a positive effect on the model performance, as a conditional statement is a specific grammatical structure (e.g., it often contains an adverbial clause) and the classifier can learn conditional specific patterns in the grammatical structure of the training instances. The fixed token length was also increased to 384 tokens.

BERT$_{Base}$: If the process fails, an error message is shown.
BERT$_{POS}$: If_SCONJ the_DET process_NOUN fails_VERB ,_PUNCT an_DET error_NOUN message_NOUN is_AUX shown_VERB ._PUNCT
BERT$_{DEP}$: If_mark the_det process_nsubj fails_advcl ,_punct an_det error_compound message_nsubjpass is_auxpass shown_ROOT ._punct

Figure 6.1: Input Sequences Used for Our Different BERT Fine Tuning Models. DEP Tags Are Marked **Orange** and POS Tags Are Marked **Blue**.

6.3 Evaluation

This section reports on the results of our experiments, in which we compare the performance of the individual methods.

Evaluation Procedure Our labeled data set is imbalanced as only 28.1 % are positive samples. In other words, only 4,215 out of the 14,983 labeled sentences contain a conditional statement. To avoid the class imbalance problem, we apply *Random Under Sampling* (see Figure 6.2). We randomly select sentences from the majority class and exclude them from the data set until a balanced distribution is achieved. Our final data set consists of 8,430 sentences with an equal distribution of sentences containing conditionals (4,215) and sentences without conditionals (4,215). We follow the idea of *Cross Validation* and divide the data set into a training, validation, and test set. The training set is used for fitting the algorithm while the validation set is used to tune its parameters. The test set is utilized for the evaluation of the algorithm based on real-world unseen data. We opt for a 10-fold *Cross Validation* as a number of studies have shown that a model that has been trained this way demonstrates low bias and variance [300]. We use standard metrics, for evaluating our approaches: Accuracy, Precision, Recall, and F_1 score [300].

When interpreting evaluation metrics, it is important to consider which misclassification (*False Negative* or *False Positive*) matters most resp. causes the highest costs. Since conditional detection is supposed to be the first step toward automatic conditional extraction, we favor Recall over Precision. A high Recall corresponds to a greater degree of automation of conditional extraction because it is easier for users to discard *False Positives* than to manually detect *False Negatives*. Consequently, we seek high Recall to minimize the risk of missed conditionals and acceptable Precision to ensure that users are not overwhelmed by *False Positives*.

Experimental Results Table 6.1 demonstrates the inability of the baseline approach to distinguish between sentences that do contain conditionals (F_1 score: 66 %) and sentences that do not (F_1 score: 64 %). This coincides with our observation from the first empirical study (see Chapter 4) that searching for cue phrases is not suitable for conditional detection. In comparison, most ML-based approaches (except KNN and DT) show a better performance. The best performance in this category is achieved by RF with an Accuracy of 78 % (gain of 13 % compared to baseline approach). The overall best classification results are achieved by our DL-based approaches. All three variants were trained with the hyperparameters recommended by Devlin et al. [131]. Even the

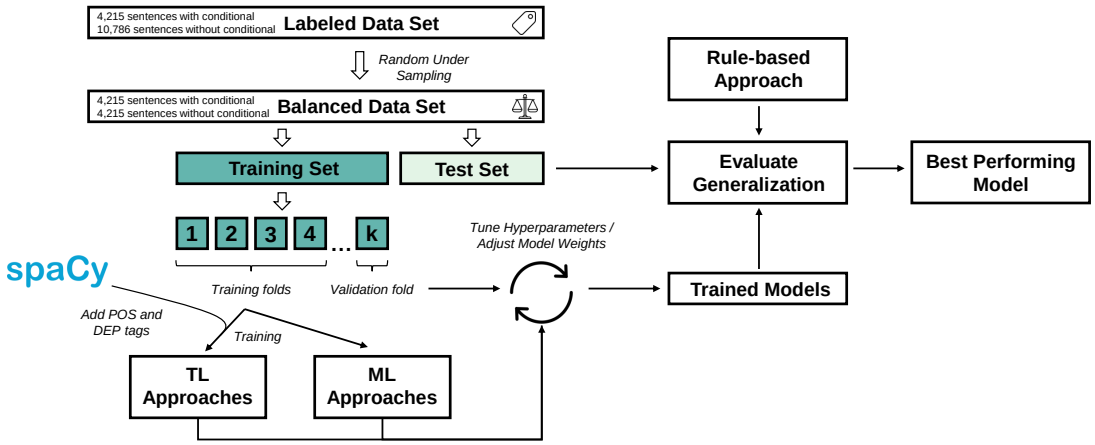


Figure 6.2: Implementation and Evaluation Procedure of Our Binary Classifier.

vanilla $\mathbf{BERT_{Base}}$ model shows a great performance in both classes (F_1 score ≥ 80 % for Conditional Present and Conditional Not Present). Interestingly, enriching the input sequences with syntactic information did not result in a significant performance boost. $\mathbf{BERT_{POS}}$ even has a slightly worse Accuracy value of 78 % (difference of 2 % compared to $\mathbf{BERT_{Base}}$). An improvement of the performance can be observed in the case of $\mathbf{BERT_{DEP}}$, which has the best F_1 score for both classes among all the other approaches and also achieves the highest Accuracy value of 82 %. Compared to the rule-based and ML-based approaches, $\mathbf{BERT_{DEP}}$ yields an average gain of 11.06 % in macro-Recall and 11.43 % in macro-Precision. Interesting is a comparison with $\mathbf{BERT_{Base}}$. $\mathbf{BERT_{DEP}}$ shows better values across all metrics, but the difference is only marginal. This indicates that $\mathbf{BERT_{Base}}$ already has a deep language understanding due to its pre-training and therefore can be tuned well for conditional detection without much further input. However, over all five runs, the use of the DEP tags shows a small but not negligible performance gain - especially regarding our main decision criterion: the Recall value (85 % for Conditional Present and 79 % for Conditional Not Present). Therefore, we choose $\mathbf{BERT_{DEP}}$ as our final approach for the detection of conditionals in RE artifacts, and include it into the `CiRA` pipeline.

Summary of Evaluation:

Our evaluation demonstrates that rule-based approaches are not capable of reliably detecting conditionals in natural language due to the ambiguity of cue phrases such as *"if"* and *"when"*. Contrary, we found that syntactically enriched BERT embeddings combined with a softmax classifier achieve a macro-F_1 score of 82 % on real-world data and outperform related approaches with an average gain of 11.06 % in macro-Recall and 11.43 % in macro-Precision.

Table 6.1: Recall, Precision, F_1 Scores (per Class) and Accuracy. We Report the Averaged Scores Over Five Repetitions and Highlight in **Bold** the Best Results for Each Metric.

			Conditional Present (Support: 435)			Conditional Not Present (Support: 408)			
		Best hyperparameters	Recall	Precision	F_1	Recall	Precision	F_1	Accuracy
Rule-based		-	0.65	0.66	0.66	0.65	0.63	0.64	0.65
ML based	NB	alpha: 1, fit_prior: True, embed: BoW	0.71	0.7	0.71	0.68	0.69	0.69	0.7
	SVM	C: 50, gamma: 0.001, kernel: rbf, embed: BoW	0.68	0.8	0.73	0.82	0.71	0.76	0.75
	RF	criterion: entropy, max_features: auto, n_estimators: 500, embed: BoW	0.72	**0.82**	0.77	**0.84**	0.74	0.79	0.78
	DT	criterion: gini, max_features: auto, splitter: random, embed: TF-IDF	0.65	0.68	0.66	0.67	0.65	0.66	0.66
	LR	C: 1, solver: liblinear, embed: TF-IDF	0.71	0.78	0.74	0.79	0.72	0.75	0.75
	AB	algorithm: SAMME.R, n_estimators: 200, embed: BoW	0.67	0.78	0.72	0.8	0.7	0.75	0.74
	KNN	algorithm: ball_tree, n_neighbors: 20, weights: distance, embed: TF-IDF	0.61	0.68	0.64	0.7	0.63	0.66	0.65
TL based	$BERT_{Base}$	batch_size: 16, learning_rate: 2e-05, weight_decay: 0.01, optimizer: AdamW	0.83	0.80	0.82	0.78	0.82	0.80	0.81
	$BERT_{POS}$		0.82	0.76	0.79	0.71	0.83	0.77	0.78
	$BERT_{DEP}$		**0.85**	0.81	**0.83**	0.79	**0.84**	**0.81**	**0.82**

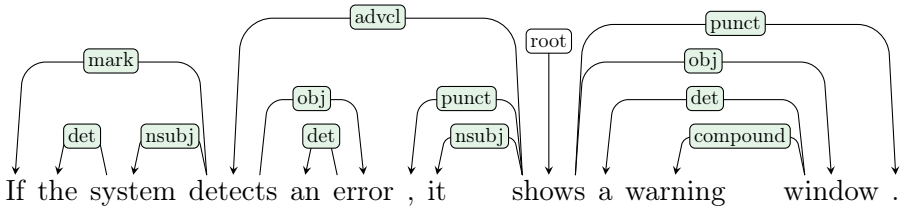
Chapter 7

Automatic Extraction of Conditionals From Requirements Artifacts

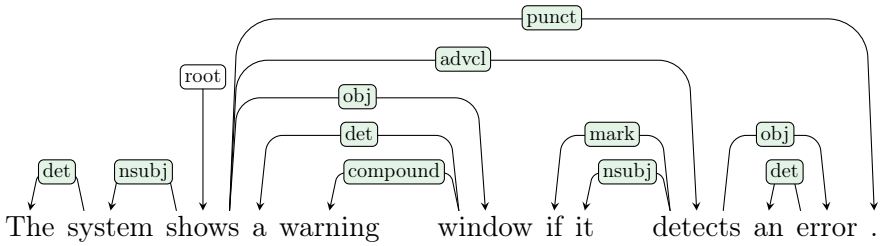
Common Thread This chapter focuses on the second step in the CiRA pipeline: the fine-grained extraction of conditionals from NL sentences. To this end, we introduce three approaches for conditional extraction and evaluate their performance on real-world data. In Section 7.1, we show how *Dependency Parsing* can be utilized to extract conditionals from NL. Our second approach builds on the idea that conditionals represent recursive structures, and uses a *Recursive Neural Tensor Network* to extract them in fine-grained form (see Section 7.2). In Section 7.3, we highlight how the extraction of conditionals can be solved by using multi-class and multi-label classifiers. We select the approach with the best performance as our final conditional extractor and incorporate it into the CiRA pipeline.

Contribution Our *Dependency Parsing*-based approach works mostly in the case of grammatically correct sentences. However, the lack of robustness against grammatical errors as well as the reliance on the performance of the dependency parser and the quality of the predefined patterns prevent its practical use. Our RNTN-based extraction approach performs better in handling grammatical errors but struggles to process *Out-Of-Vocabulary* words. Specifically, the RNTN is unsure to which segments the unknown tokens should be assigned (e.g., does the token belong to an *antecedent* or *consequent*?) and fails to understand the semantics of the sentence. Contrary, we found that a sigmoid classifier built on RoBERTa embeddings is more robust and best suited to extract conditionals in fine-grained form. It achieves a macro-F_1 score of 86 % when evaluated on a real-world data set of 1,946 sentences.

Related Publications This chapter is taken, directly or with minor modifications, from previous publications [1, 6, 8, 10].



(a) Dependency Tree Representing REQ 1.



(b) Dependency Tree Representing REQ 2.

Figure 7.1: Independence of Dependency Parsers From Word Order in NL.

7.1 Extracting Conditionals by Dependency Parsing

This section introduces our first approach for extracting conditional statements from NL. In Section 7.1.1, we describe the principal idea behind our approach. Section 7.1.2 presents a new formal language designed to define specific *antecedent-consequent*-patterns that are used as the basis for subsequent extraction. Finally, Section 7.1.3 presents a case study demonstrating the feasibility of our approach in cooperation with two industry partners: *Leopold Kostal GmbH & Co. KG* (automotive) and *Allianz Deutschland AG* (insurance).

7.1.1 Principal Idea

In the past, rule-based approaches were primarily used for the extraction of information. Such approaches are usually easy to implement (e.g., by using regular expressions) and are therefore a first quick solution to extract specific content from texts. Rule-based systems, however, can quickly become difficult to maintain, since a series of different rules needs to be defined to handle complex scenarios. They depend on the word order causing their application to be inflexible. This can be demonstrated by two simple requirements. REQ 1 states that *"If the system detects an error, it shows a warning window"* and REQ 2 describes the system behavior as follows: *"The system shows a warning window if it detects an error"*. Both requirements specify the same system behavior and are therefore semantically identical. However, they possess a different word order, so that two different syntax rules are required for the extraction of the *antecedent* (*"detection of an error"*) and *consequent* (*"showing a warning window"*). This causes unnecessary extra work and results in an inflexible overall solution.

The core idea behind our approach lies in the use of a dependency parser. Irrespective of the word order, the parser describes the content of a sentence by analyzing the relationships between its words. These relationships can be depicted as a tree representing the overall structure of the sentence. In this way, considered requirements share the

same dependency tree although they have a different syntax (see Figure 7.1). The dependency parser identified that the token *"shows"* is the root node of the dependency tree for REQ 1 (see Figure 7.1a) as well as REQ 2 (see Figure 7.1b). In both cases, the relationship of *antecedent* and *consequent* is marked by the $advcl(shows, detects)$ relation. Except for the position of the marker *"if"*, both dependency trees are identical. Interpreting and fragmenting this tree into *antecedents* and *consequents* would thus cover two different formulation styles of the same system functionality. We build on the fact that a dependency tree consists of a set of subtrees, which represent different parts of the parsed sentence. We aim to identify those subtrees that comprise the individual *antecedents* and *consequents*. For this purpose, we traverse the tree and identify individual subtrees by pattern matching. By traversing any kind of dependency tree and applying a defined set of patterns on its nodes, our approach allows us to flexibly detect and extract *antecedent-consequent*-relationships within NL sentences regardless of their syntax. We hypothesize that the dependency tree matching requires substantially fewer patterns compared to formulating the patterns specifically on the syntax.

7.1.2 Implementation

This section specifies in detail how our approach works. We first design a formal language for defining patterns that are used by our approach to search a dependency tree for *antecedents* and *consequents* (see Section 7.1.2.1). We describe the application of these patterns using a running example to illustrate the operation of our approach (see Section 7.1.2.2).

7.1.2.1 A New Formal Language for Dependency Pattern Matching

A dependency tree D is a directed rooted tree with words as nodes and labeled relations as edges. D can be represented as a tuple (V, R) where:

- V is a finite set of nodes. Each node $v \in V$ represents one token (word or punctuation) of a sentence.
- R is a finite set of edges between two nodes. Each edge $r \in R$ represents a *Dependency Relation* $\rightarrow$ between two words with a dependency label d. A *Dependency Relation* can be expressed as a triple $d(v_i, v_j)$.

At each node $v_i \in V$, we can define a separate subtree D_i where v_i is the root of D_i. We define $D_i(V_i, R_i)$ as a subtree of $D(V, R)$ if $V_i = \{v_j \in V \mid v_i \rightarrow^* v_j\}$, where $\rightarrow^*$ denotes a subordinate relation. Accordingly, the subtree contains only those nodes that depend on root v_i. Based on these mathematical definitions, we created a formal language that allows specifying subtrees and their properties as patterns. A pattern p_i complies with the following grammar, where $\vDash$ signifies produces, | is an or operator, <name> are production names.

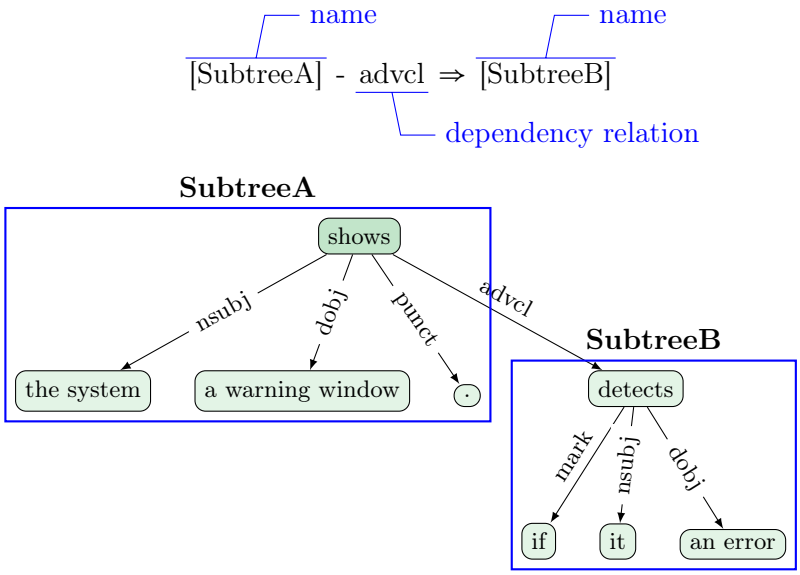
$$\langle \text{pattern} \rangle \vDash \big([\langle \text{e1} \rangle] \mid \langle \text{e2} \rangle \big) \text{ - } \langle \text{e3} \rangle \Rightarrow \big(\langle \text{pattern} \rangle \mid [\langle \text{e1} \rangle] \mid \langle \text{e2} \rangle \big) \tag{7.1}$$

$$\langle \text{e1} \rangle \vDash \texttt{name} \tag{7.2}$$

$$\langle \text{e2} \rangle \vDash \texttt{POS tag: word} \tag{7.3}$$

$$\langle \text{e3} \rangle \vDash \texttt{dependency relation} \tag{7.4}$$

Accordingly, a pattern $p_i = \{e_i, ..., e_n\}$ can be constructed by combining three different elements e_i. Each pattern may be applied to any node $v_i \in V$. The element e_1 can be used to specify the desired name of a subtree D_i with root v_i (see Equation 7.2). In order to check v_i for a specific word and its corresponding POS tag, e_2 can be used (see Equation 7.3). By e_3 a certain label of a *Dependency Relation* can be defined with v_i as the head (see Equation 7.4). The terminal ⇒ indicates a move from v_i to the next subordinate node v_j. Formally, this node is the dependent in the triple $d(v_i, v_j)$ defined by e_3. The expression preceded by ⇒ either specifies the beginning of a new pattern, a new subtree D_j or can be used to check the POS tag of v_j and its corresponding value. Let us assume the following example, which is applied on the root of $D(V, R)$:



The pattern is read from left to right and specifies the two subtrees *"SubtreeA"* and *"SubtreeB"*. To match the pattern, the *Dependency Relation* $advcl(v_i, v_j)$ must exist. Since it exists in the present example, *"SubtreeA"* will be created as D_{shows} with root v_{shows} and *"SubtreeB"* as $D_{detects}$ with root $v_{detects}$. Finally, to extract the two subtrees from D and separate them from each other, $advcl(v_{shows}, v_{detects})$ is deleted. By means of such a pattern you can describe which parts should belong to a subtree and which not. Hence, this pattern can be expressed as follows: *"SubtreeA"* should contain all subordinate nodes and edges of v_i, only the part beneath the *Dependency Relation advcl* should belong to *"SubtreeB"*. In some cases, a group of patterns must be applied to define a complex subtree (see subtree *"PartA"* in pattern group 7 in Table 7.1). For this purpose, we define P_i as a pattern group, which consists of multiple rows, each row being a separate pattern. A pattern group can be expressed as $P_i = \{p_i, ..., p_n, \perp\}$, where $\perp$ denotes the end of the pattern as a terminal symbol (i.e., end of the last row). The sum of all pattern groups is expressed as $P = \{P_i, ..., P_n\}$.

7.1.2.2 Pattern Matching of the Dependency Tree

Utilizing the introduced language, we created a set of patterns in both English and German that cover different *antecedent-consequent*-relationships. In total, we created 38

Table 7.1: Excerpt of Developed Pattern Groups.

#	Pattern Group
1	[Consequent] - advcl ⇒ [Antecedent] - mark ⇒ IN:'if' [Consequent] - advmod ⇒ RB: 'then'
2	[Consequent] - advcl ⇒ [Antecedent] - mark ⇒ IN:'if | because | although'
3	[Antecedent] - ccomp ⇒ [Consequent] [Antecedent] - advmod ⇒ WRB:'when' [Antecedent] - advmod ⇒ RB: 'then'
4	[Antecedent] - amod ⇒ JJ:'due' - prep ⇒ TO:'to' - pobj ⇒ [Consequent]
6	VBN:'provided' - ccomp ⇒ [Antecedent] - dobj ⇒ [Consequent] [Antecedent] - complm ⇒ IN:'that'
7	[PartA] - preconj ⇒ CC:'neither' [PartA] - cc ⇒ CC: 'nor' [PartA] - conj ⇒ [PartB]
8	[PartA] - cc ⇒ CC:'and' [PartA] - conj ⇒ [PartB] [PartA] - preconj ⇒ DT:'both'
9	[PartA] - cc ⇒ CC:'or' [PartA] - conj ⇒ [PartB]
10	[Condition] - nsubj ⇒ [Variable]

English and 43 German patterns. Table 7.1 depicts an excerpt from the set of formulated patterns. For all other patterns please refer to our *Github* repository.[1] We apply the amount of patterns on the dependency tree to extract the respective *antecedents* and *consequents*. For this purpose, we implemented an algorithm that traverses a dependency tree starting from its root and applies the patterns to the individual nodes of the tree. Its exact operation is described in Algorithm 2. For better comprehensibility of the algorithm we will apply it to the given requirement:

> *"If the customer is traveling with a parent, or the customer is older than 23 years and the customer shows a valid driving license, the system does not charge an increased fee."*

As a first step, the requirement is converted into a dependency tree (see Figure 7.2). Subsequently, we traverse the created dependency tree in order to extract the *antecedents* and *consequents*.

7.1.2.3 Procedure of the Algorithm

The algorithm begins at the root of the dependency tree and starts matching the defined patterns. If a pattern matches, the algorithm splits the dependency tree into the subtrees specified by the pattern. Subsequently, the algorithm applies all patterns again to the newly created subtrees with the aim of successively decomposing them into smaller

1 Available as *Open Source* at `https://github.com/qualicen/specmate`. The class `PatternbasedCEGGenerator` is recommended as a starting point to comprehend the approach presented in this section (see bundle `specmate-model-generation`).

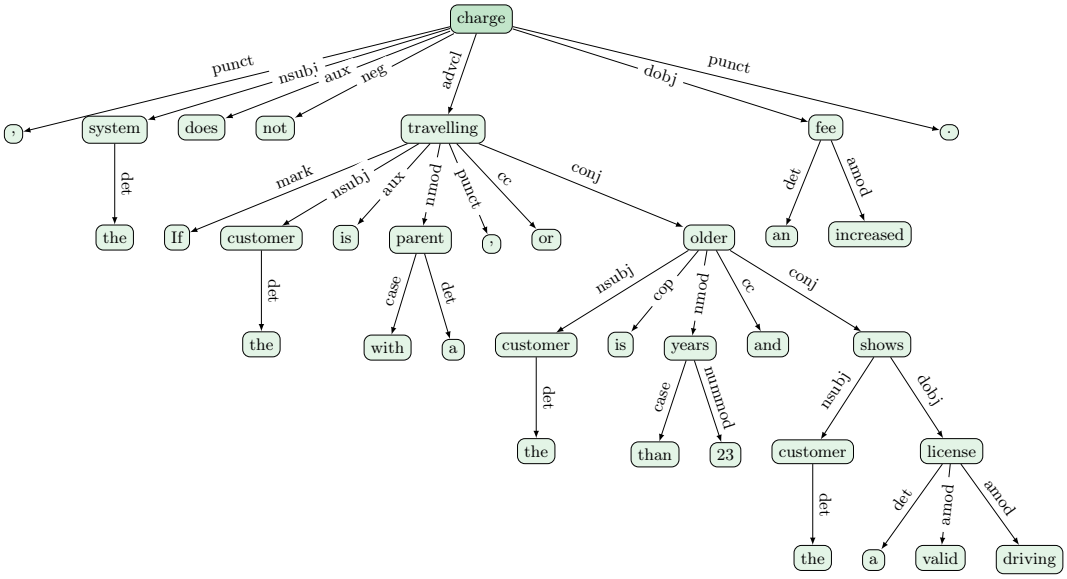
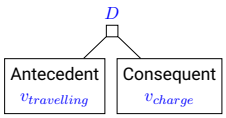


Figure 7.2: Dependency Tree Representing the Requirement *"If the customer is traveling with a parent, or the customer is older than 23 years and the customer shows a valid driving license, the system does not charge an increased fee."*

subtrees. This recursive process is performed until all subtrees are broken down into conditions and variables. If no pattern matches in any of the subtrees, the algorithm terminates. While traversing the dependency tree, the algorithm creates a *Subtree Syntax Tree* (SST) to reflect the relationship between the subtrees. This tree contains the names of the respective subtrees, their root nodes as well as a list of their child nodes. To visualize the relations (conjunction, disjunction, etc.) between the subtrees, the algorithm uses labeled edges indicating the type of connection. Therefore, every edge in the SST gets a $type \in \{and, nor, xor, or\}$. The depth of the tree corresponds to the number of algorithm recursions. In the following we illustrate the algorithm by extracting *antecedents* and *consequents* from the dependency tree shown in Figure 7.2. After each recursion, we give an insight into the successive construction of the SST. It should be noted that the number of required recursions depends on the complexity of the dependency tree.

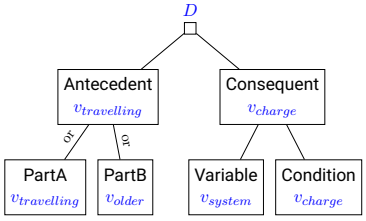
1. Recursion In the present example, the root node v_{charge} contains all information about the *consequent*, while all nodes located below the dependency edge *advcl* specify the *antecedents* (i.e., all child nodes of $v_{travelling}$). This scenario is covered by Pattern 2. According to the pattern, the algorithm checks whether the current node is in an *advcl* dependency relation to one of its child nodes. As this dependency relation exists in the present example, the algorithm deletes the dependency relation $advcl(v_{charge}, v_{travelling})$ and splits the tree into two subtrees - D_{charge} and $D_{travelling}$. The first is named *"Consequent"*, while the second subtree is called *"Antecedent"*. As a result, the first iteration returns $V_{charge} = \{v_{the}, v_{system}, v_{does}, v_{not}, v_{charge}, v_{an}, v_{increased}, v_{fee}\}$. All other nodes $v \in V$ are contained in $V_{travelling}$ except v_{if}, since the algorithm also deletes the dependency relation $mark(v_{travelling}, v_{if})$ according to Pattern 2. Nodes containing only punctuation (e.g. $v_.$) do not provide useful information for the *consequent* or *antecedent* and are therefore neglected by the algorithm.

Output of 1. Recursion



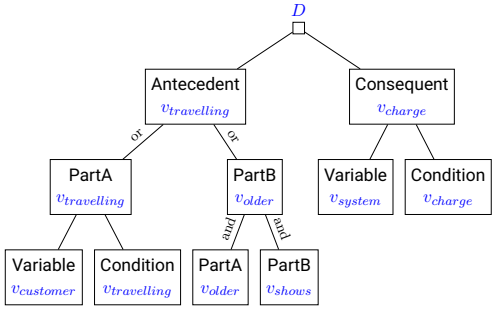
2. Recursion During the second iteration of the algorithm, each rule is applied to the roots of the two subtrees *"Antecedent"* and *"Consequent"*. v_{charge} matches Pattern 10, while $v_{travelling}$ matches Pattern 9. Pattern 10 splits the subtree *"Consequent"* into two further subtrees *"Condition"* and *"Variable"* - D_{charge} and D_{system}. After applying the pattern on v_{charge}, V_{system} includes v_{the} and v_{system} specifying the variable of the *consequent*, whereas $V_{charge} = \{v_{does}, v_{not}, v_{charge}, v_{an}, v_{increased}, v_{fee}\}$ defining the condition of the *consequent*. The first part of Pattern 9 is matched by $cc(v_{travelling}, v_{or})$ followed by the fact that the word *"or"* has been classified with POS tag CC. The second part leads to a splitting of $D_{travelling}$ into the two subtrees *"PartA"* as $D_{travelling}$ with $v_{travelling}$ as root and *"PartB"* as D_{older} with v_{older} as root. As a result, the second iteration returns $V_{travelling} = \{v_{the}, v_{customer}, v_{is}, v_{travelling}, v_{with}, v_a, v_{parent}\}$. V_{older} contains all child nodes of v_{older}.

Output of 2. Recursion



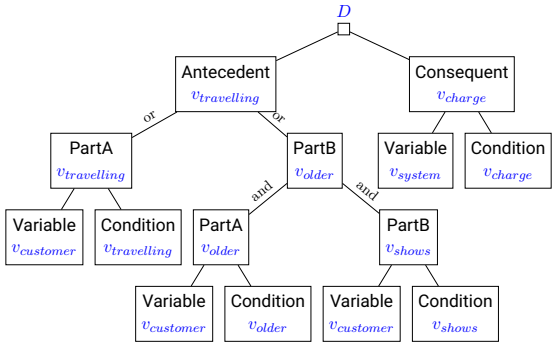
3. Recursion In the third iteration, no pattern matches the subtrees D_{charge} and D_{system} because these already exist in the form of condition resp. variable and therefore do not have to be split further. In contrast, $v_{travelling}$ matches Pattern 10, while v_{older} matches Pattern 8 as one of the conjunction patterns. $D_{travelling}$ is thus divided into the two subtrees $D_{customer}$ and $D_{travelling}$ with $V_{customer} = \{v_{the}, v_{customer}\}$ and $V_{travelling} = \{v_{is}, v_{travelling}, v_{with}, v_a, v_{parent}\}$. D_{older} contains two *antecedents* connected by a conjunction. This conjunction is resolved by Pattern 8 by dividing D_{older} into *"PartA"* as D_{older} with v_{older} as root and *"PartB"* as D_{shows} with v_{shows} as root.

Output of 3. Recursion



4. Recursion In the fourth iteration, the pattern matching is only successful for the subtrees D_{older} and D_{shows}. Both subtrees contain no further connections to other *antecedents* and are therefore divided into the subtrees *"Condition"* and *"Variable"* according to Pattern 10. In summary, our approach has thus identified three *antecedents* and one *consequent* by traversing the dependency parse tree. The combinatorics of the extracted *antecedents* and the *consequent* is illustrated by the SST below and constitutes the final output of our method.

Final Output of the Algorithm



7.1.3 Evaluation

In cooperation with *Leopold Kostal GmbH & Co. KG* (automotive) and *Allianz Deutschland AG* (insurance), we conduct a case study to evaluate whether our *Dependency Parsing*-based approach is capable of extracting conditionals from real-world data. For our study, we follow the guidelines by Runeson and Höst [269] for conducting case study research. In the following, we report on our research questions, investigated study objects, and study results.

7.1.3.1 Research Questions

For the evaluation, we are interested in the following research questions (RQ):

- **RQ 1:** Can our automated approach based on *Dependency Parsing* extract conditionals from *Natural Language* in fine-grained form?
- **RQ 2:** What are the reasons for incorrectly extracted conditionals or even completely missed conditionals?

7.1.3.2 Study Objects

Allianz Deutschland AG provided us with 72 user stories. All of these user stories contain acceptance criteria that are expressed using conditional statements and are therefore suited for assessing our approach. The examined user stories contain 259 acceptance criteria and are written in English. *Leopold Kostal GmbH & Co. KG* provided us with a functional specification including 255 requirements. Out of these 255 requirements, we consider 79 requirements for our case study since they describe a functional behavior by conditional statements. All investigated requirements are written in English.

Algorithm 2: Recursive Pattern Matching Algorithm for Fine-Grained Conditional Extraction From an Arbitrary Dependency Tree D

input : $D(V, R)$
output : List of matched P_i and D_{sub}

```
1  /* start with D_sub = {D} since the whole tree is the first subtree */
2  for D_i ∈ D_sub do
3      v_i = root of D_i /* apply all patterns on the current node v_i */
4      for P_i ∈ P do
5          for p_i ∈ P_i do
6              /* successful match of P_i */
7              if p_i =⊥ then
8                  add ∀D_name ∈ D_sub.pattern to D_sub
9                  /* build the RST */
10                 connect D_i to ∀D_name ∈ D_sub.pattern
11                 empty D_sub.pattern
12                 /* move to next subtree and apply all rules again */
13                 return P_i and go to step 1
14             else
15                 for e_i ∈ p_i do
16                     if e_i = [name] then
17                         /* P_i specifies a subtree by multiple sub patterns */
18                         if D_name ∈ D_sub.pattern then
19                             set v_i = root of D_name
20                         else
21                             create D_name
22                             set root of D_name = v_i
23                             /* each matching P_i gets its own list of subtrees */
24                             add D_name to D_sub.pattern
25                     else if e_i = dependency ⇒ then
26                         check dependency(v_i, v_j) ∈ {v_i → v_j ∈ R_i}
27                         if true then
28                             delete dependency(v_i, v_j)
29                             /* move to dependent */
30                             set v_i = v_j
31                             /* check the next element of p_i */
32                             continue
33                         else
34                             /* non-matching P_i */
35                             empty D_sub.pattern and go to step 3
36                     else if e_i = POS Tag: value then
37                         check v_i matches POS Tag: value
38                         if true then
39                             continue
40                         else
41                             /* non-matching P_i */
42                             empty D_sub.pattern and go to step 3
```

7.1.3.3 Study Results

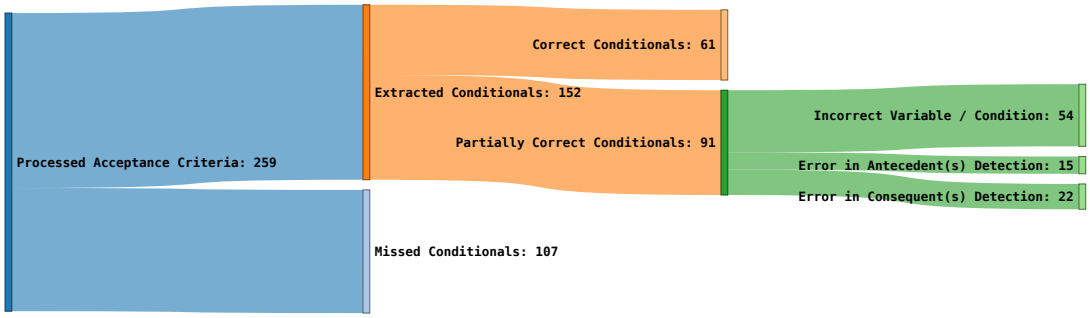
We report on the results of our study structured by our research questions. Figure 7.3 provides an overview of the performance of our approach when applied to our study objects. The vertical bars in the *Sankey* diagram correspond to certain categories (e.g., *"correctly extracted conditionals"*). The thickness of links between the bars encodes the *count* of study objects belonging to a category.

Results for RQ 1 Applying our approach to the 259 acceptance criteria provided by *Allianz* results in a set of 152 automatically extracted conditionals (see Figure 7.3a). Hence, our approach failed to process 107 acceptance criteria and extract their included conditionals (41.3 %). When analyzing the automatically extracted conditionals, we found that only 61 conditionals are *correct* (i.e., our approach identified all *antecedents* and *consequents* as well as their corresponding variables and conditions, and considered conjunctions and disjunctions). However, the majority (91) of the extracted conditionals were not entirely correct. In case of 54 conditionals, the *antecedents* and *consequents* were detected properly, but the variables and conditions were assigned incorrectly. In other words, the approach assigned some tokens to the wrong subtrees at the lower levels. In the case of 15 of the partially correct conditionals, certain *antecedents* were overlooked. Similarly, *consequents* were missed during the extraction of 22 of the conditionals. In these cases, the approach failed to interpret conjunctions / disjunctions and ignored the combinatorics of multiple *antecedents* or *consequents*.

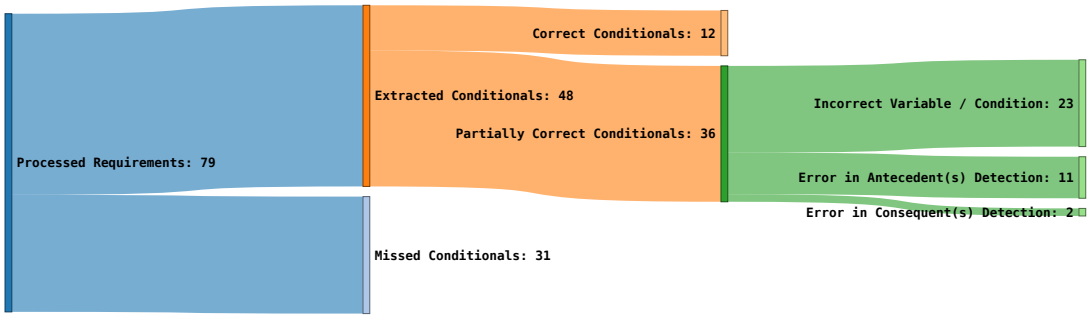
Our approach shows a similar performance when applied to the data provided by *Kostal* (see Figure 7.3b). Out of the 79 requirements, our approach extracts 48 conditionals. It thus fails in the processing of 11 requirements and is unable to extract their contained conditionals (39.2 %). We observed that 12 of the extracted conditionals were correct. The majority (36) of the extracted conditionals, however, were only partially correct processed by our approach. In case of 23 conditionals, the variables and conditions of the *antecedents* and *consequents* were assigned incorrectly. We notice that *antecedents* were missed during the extraction of 11 conditionals. In the case of two of the partially correct conditionals, certain *consequents* were overlooked.

The similar performance of our approach on both data sets is visually underlined by the *Sankey* diagrams (see Figure 7.3). The thickness of the links highlights that the proportions of study objects in the investigated categories are similar for both cases. For example, the proportion of missed conditionals is almost identical: 41.3 % in case of *Allianz* data and 39.2 % in case of *Kostal* data. Furthermore, in both cases most of the extracted conditionals were only partially correct: 60 % (*Allianz*) and 75 % (*Kostal*).

Results for RQ 2 The main reason for the high number of missed conditionals lies in the lack of robustness of our approach to grammatical errors in a sentence. In case of, e.g. spelling mistakes and missing punctuation marks, the sentences can not be parsed correctly, whereby a wrong dependency tree is created causing our algorithm to fail. This poses a threat to the applicability of our approach in practice since requirements often suffer from poor quality. We also encountered problems with handling complex variable names, since they were not included in the vocabulary on which the dependency parser was trained. In some cases, the input sentence was grammatically correct, but the parser produced a wrong dependency tree from which our approach derived a wrong



(a) Extracted Conditionals From *Allianz* Data.



(b) Extracted Conditionals From *Kostal* Data.

Figure 7.3: Evaluation Results of Our Conditional Extraction Approach Based on *Dependency Parsing*.

conditional. This emphasizes again the dependence of our approach on the output of the used dependency parser: as soon as an error occurs during the creation of the dependency tree, our approach fails. Additionally, our approach is dependent on the manually defined *antecedent-consequent*-patterns. In the case study, our approach was not able to properly process all generated (correct) dependency trees because specific rules were missing in our pattern set. The extraction capability of our approach is thus strongly influenced by the quality and completeness of the predefined patterns.

Summary of Evaluation:

In conclusion, the presented approach is not suitable for a fully automated conditional extraction from NL. Across all 338 study objects, it could not extract 138 conditionals (error rate of 40.82 %). In total, however, the approach was able to extract 73 conditionals completely correct and 127 conditionals partially correct. Our study proves that the idea of using *Dependency Parsing* for conditional extraction is generally reasonable and indeed works in the case of grammatically correct sentences. However, the lack of robustness against grammatical errors as well as the reliance on the performance of the dependency parser and the quality of the manually crafted patterns prevent the practical use of our approach.

7.2 Extracting Conditionals by Recursive Neural Tensor Networks

This section introduces our second approach for extracting conditional statements from *Natural Language*. In Section 7.2.1, we describe the principal idea behind using an RNTN for conditional extraction and explain the key differences compared to our first approach. Further, we detail the creation of an appropriate corpus of binary trees suitable for training our approach (see Section 7.2.2). Section 7.2.3 outlines the functionality and implementation of our conditional approach based on an RNTN. We evaluate our approach in Section 7.2.4.

7.2.1 Principal Idea

An RNTN is based on the idea that *Natural Language* can be understood as a recursive structure [128]. For example, the syntax of a sentence is recursively structured, with noun phrases containing relative phases, which in turn contain further noun phrases, and so on. An RNTN is capable of recovering this recursive structure and helps to better understand the composition of a sentence. We argue that a conditional statement also represents a recursive structure as it consists of *antecedents* and *consequents*, which in case of conjunctions and disjunctions consists of further *antecedents* and *consequents*, and so on [2]. This results in a tree-like structure of *antecedent* and *consequent* nodes forming the full sentence. By recovering this tree-like structure, we do not lose the combinatorics between the *antecedents* and *consequents* which allows us to entirely extract the conditional. Furthermore, it allows us to split single sentence fragments into increasingly smaller parts (e.g., *antecedents* can be split into variables and conditions, which in turn can be decomposed into further more granular text fragments). In this way, we enable a fine-grained conditional extraction. Utilizing an RNTN for conditional extraction has several advantages compared to our *Dependency Parsing*-based approach:

- **Increased robustness against grammatical errors:** Our *Dependency Parsing*-based approach generally fails and does not produce any output in case of grammatical errors (i.e., it does not even extract parts of the conditional but aborts completely). Our RNTN-based approach is much more robust against grammatical errors and does not depend on a pre-processing step like *Dependency Parsing*. Rather, the RNTN creates the tree structure in a bottom-up fashion and decides from adjacent to adjacent pair which segments semantically belong together and should be merged accordingly. Grammatical errors (e.g., wrong punctuation, spelling mistakes) thus only affect individual segments in the sentence and do not negatively influence the entire sentence comprehension of the RNTN. Naturally, the generated tree-structure may contain isolated errors, but the RNTN-based approach does not fail and produces an (albeit only partially correct) output.

- **No dependence on predefined patterns:** Unlike our *Dependency Parsing*-based approach, the RNTN does not rely on handcrafted patterns but acquires its language understanding during the training process. The extraction capability is therefore not dependent on the quality and completeness of a given rule set. It should be stressed, however, that the RNTN-based approach also involves manual work, since a training corpus must first be manually annotated in order to allow

supervised learning. Thus, the performance of the RNTN depends on the quality of the annotated data.

7.2.2 Training Corpus Creation

To the best of our knowledge, we are the first to utilize an RNTN for RE purposes. Accordingly, there is no labeled corpus of requirements available in the RE community that could be used to train an RNTN. This section describes how we created a suitable training corpus.

7.2.2.1 Data Collection

Since we train the RNTN to extract conditional statements, we need requirements that include conditionals. Hence, we searched for conditionals in our gold standard corpus of 212,186 requirements presented in Chapter 4. Since manual analysis of all sentences is not practicable, we searched for specific cue phrases that usually indicate conditionals [4]. We focused on the cue phrases: *"if"*, *"when"*, *"in order to"*, *"due to"*, *"because"*, *"since"* and *"in (the) case of"*. We randomly searched the data set for sentences containing these cue phrases and discussed in the research group whether a sentence (1) represents a functional requirement and (2) includes a conditional. We continued the search until we reached a reasonable number of sentences for the training of the RNTN. As a result, the filtered data set consists of 1,571 requirements that include a conditional statement.

7.2.2.2 Annotation Labels

We need to indicate specific segments in the requirements so that the RNTN can learn to identify the structure of a conditional. Since we aim to extract conditionals in fine-grained form, we annotated the 1,571 sentences with 27 different segments. To minimize the annotation effort, we analyzed which segments actually need to be annotated manually and which segments can be labeled automatically. We found that 12 of the 27 segments can be assigned automatically. For example, Word and Punct segments can be rule-based annotated, since they only occur at the lowest level of the tree.

- **Manually assigned labels:**
  1. Variable This label indicates noun phrases in a sentence.
  2. Condition This label indicates the verb phrases that belong to a variable.
  3. Negation This label indicates negations (e.g., negated conditions).
  4. Statement This label indicates combinations of variable and condition segments.
  5. Not Relevant (NR) This label indicates segments that are not part of a conditional statement.
  6. Key C This label highlights certain cue phrases that indicate conditionals.
  7. Antecedent This label indicates an *antecedent* segment.
  8. Consequent This label indicates a *consequent* segment.
  9. Conditional Statement This label captures all segments that belong to the conditional statement.
  10. And This label is used to annotate conjunctions.

11. Or This label is used to annotate disjunctions.
12. Root Sentence This label always represents the root node of a sentence.
13. Key NR This label highlights certain cue phrases that indicate not-relevant segments in a sentence.
14. Insertion This label is used to annotate any kinds of insertions in a sentence (e.g., bracket expressions that are not essential for the interpretation of a sentence but provide additional information).
15. Sentence This label is used to connect, e.g., not-relevant segments and segments that belong to a conditional of a sentence. Contrary to the Root Sentence segment, it does not contain the ending punctuation mark of the sentence.

- **Automatically assigned labels:**

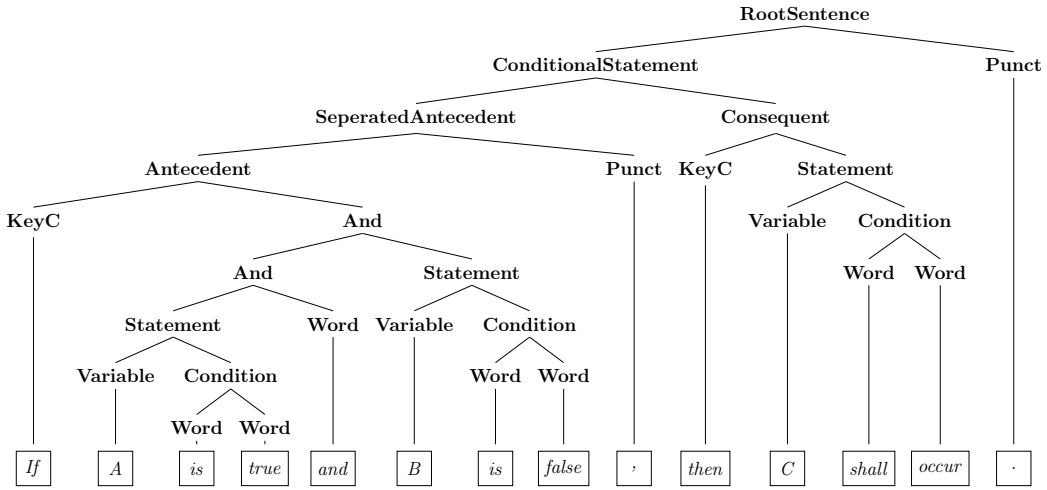
1-9. Separated... {Antecedent | Statement | And | Conditional Statement | Negation | Not-relevant | Or | Consequent | Variable}: This label is used to highlight self-contained text fragments that are syntactically separated from other fragments. The label is needed, for example, when a Statement segment will be merged with a comma token. The comma turns the segment into a Separated Statement.
10. Word This label is distributed at the bottom level of the tree and assigned to the individual words of a sentence.
11. Punct This label is used to indicate punctuation marks.
12. Symbol This label is used to mark special symbols.

7.2.2.3 Annotation Procedure

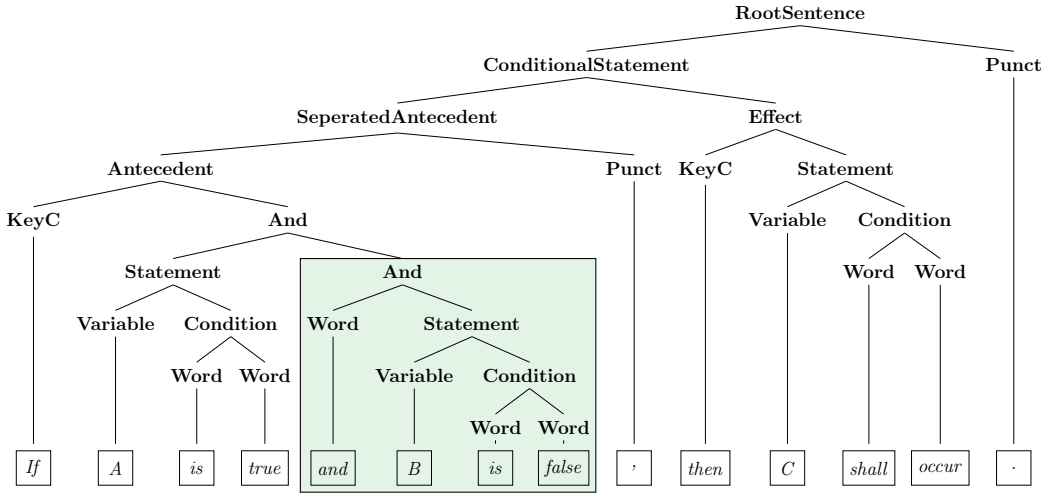
As described in Section 2.3.2, an RNTN builds the tree-structure in a bottom-up fashion. In order to reflect the composition of a sentence, it tries to identify tokens that belong contextually together and merges them into a segment. When adding new tokens to a segment, it decides what information is added by the new token and whether the label of the segment needs to be adjusted or not. We considered this approach during the annotation process and wrote an annotation guideline specifying five steps according to which the sentences should be labeled. We involved four annotators and conducted a workshop where we discussed several examples. Since the quality of the annotations is fundamental for the performance of our final model, we describe the applied annotation procedure in detail. We use the following requirement as our sentence to be annotated (see Figure 7.4): *"If A is true and B is false, then C shall occur."*

Step 1: Identify Words, Punctuation Marks, and Special Symbols. The first annotation level is trivial. Each individual word is assigned the label Word, while the punctuation marks are annotated with Punct. The sentence does not contain any special symbols.

Step 2: Identify Variables and Conditions. Also, Examine for Negations. On the second level, we distinguish between variables and conditions. In the present case, *"A"*, *"B"* and *"C"* can be marked as a Variable segment, while the verb phrases are labeled as Condition segments. None of the conditions is negated.



(a) Left Branching.



(b) Right Branching.

Figure 7.4: Correct Binary Parse of the Requirement: *"If A is true and B is false, then C shall occur."* Segments Are Highlighted in **Bold**. Tokens Are Indicated by Frames. Upper Figure: Parse Tree After Applying Left-Branching. Lower Figure: Parse Tree After Applying Right-Branching. Green Highlighting Demonstrates the Difference Between Both Branching Methods.

Step 3: Identify Statements and Understand Combinatorics. Most challenging is the annotation of the third level, which can be illustrated by the adjacent pair [A, is true]. According to the second annotation level, *"A"* represents a Variable segment, while *"is true"* is part of a Condition segment. Combined, the two segments form the expression *"A is true"*. As readers of the sentence, we know due to the preceding *"if"* phrase that this expression represents a cause and should be annotated with a corresponding label. However, since the RNTN builds the tree in a bottom-up fashion, it is unable to take the cue phrase into account when merging [A, is true]. The content of both segments does not allow any conclusion about the presence of an *antecedent*. In fact, both segments could also be part of an Consequent segment or even part of a Not-relevant segment. However, the combination of a noun phrase and verb phrase allows us to infer a Statement segment. Similarly, the adjacent pair [B, is false] can be also annotated as a Statement. To cover the combinatorics between both statements, they must be connected by a conjunction. However, this is not directly possible, because both segments are not adjacent and are interrupted by an *"and"* token. This results in two options: We can either merge *"and"* with the left neighbor (left branching, see Figure 7.4a) or with the right neighbor (right branching, see Figure 7.4b). In our paper, we experimented with both branching methods and implemented them in our exporter (see lines 24 - 38 in Algorithm 3). In the case of our exemplary requirement, we assume that right branching is used. Consequently, we label the expression *"and B is false"* as an And segment, because the added *"and"* token turns the statement into a conjunctive statement (see cyan highlighting in Figure 7.4b).

Step 4: Identify *Antecedents* and *Consequents*. The RNTN can only recognize that the conjunctive statement represents an *antecedent* by merging the adjacent pair [If, A is true and B is false]. The cue phrase *"if"* provides valuable information to the model, which consequently changes the label of the segment from Statement to Antecedent. The same applies if you combine [then, C shall occur]. The cue phrase *"then"* indicates that this segment represents an Consequent.

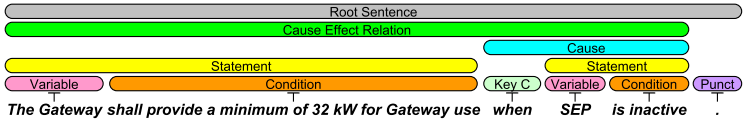
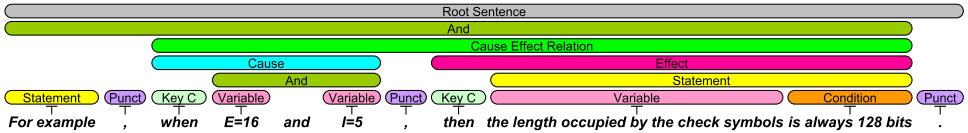
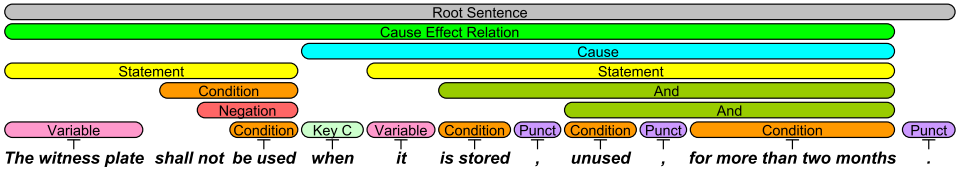
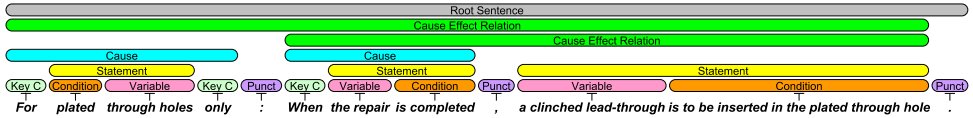
Step 5: Connect *Antecedent-Consequent*-Pairs. In the last step, the related *antecedent* and *consequent* pairs are joined into a Conditional Statement segment. The exemplary requirement contains only a single *antecedent-consequent* relation. In some cases, however, a sentence may contain several conditionals, so that several Conditional Statement segments must be annotated (see [S4] in Table 7.2).

7.2.2.4 Annotation Tool and Tree Exporter

An RNTN requires a strongly structured training input. More specifically, it needs to be trained on binary trees. This renders the annotation of individual segments laborious and error-prone. To support the annotators, we used the web-based *brat*[2] annotation platform [301]. Instead of requiring the annotators to assign separate segment labels for each adjacent token pair, we allow the annotation of segments that span multiple tokens (see Table 7.2). Consequently, the manual annotations do not need to follow a binary

2 Since *brat* was not originally designed for annotating multiple layers, we slightly modified the platform. We share the customized code in our *Github* repository: `https://github.com/springto/Fine-Grained-Causality-Extraction-From-NL-Requirements`.

Table 7.2: Examples of Manual Annotations and Their Corresponding Binary Structured Output. Segment Names Are Represented by Numbers in the Binary Annotations (g.g., a Word Segment Is Indicated by 23). The Exact Mapping of Segment Name to Corresponding Number Is Available in Our *Github* Repository.

#	Manual Annotations in *brat*	Binary Structured Annotations Generated by Our Exporter
[S1]		(1 (13 (10 (9 (23 The) (23 Gateway)) (8 (8 (8 (8 (8 (8 (8 (8 (23 shall) (23 provide)) (23 a)) (23 minimum)) (23 of)) (23 32kW)) (23 for)) (23 Gateway)) (23 use))) (11 (6 when) (10 (9 SEP) (8 (23 is) (23 inactive)))))(3 .))
[S2]		(1 (20 (20 (17 (23 For) (23 example))(3 ,)) (13 (14 (11 (6 when) (4 (4 (9 E=16) (23 and)) (9 I=5)))(3 ,)) (12 (6 then) (10 (9 (9 (9 (9 (9 (23 the) (23 length)) (23 occupied)) (23 by)) (23 the)) (23 check)) (23 symbols)) (8 (8 (8 (23 is) (23 always)) (23 1280)) (23 bits))))))(3 .))
[S3]		(1 (13 (10 (9 (9 (23 The) (23 witness)) (23 plate)) (16 (16 (23 shall) (23 not)) (8 (23 be) (23 used)))) (11 (6 when) (10 (9 it) (4 (4 (8 (23 is) (23 stored))(3 ,)) (4 (4 (8 unused)(3 ,)) (8 (8 (8 (8 (23 for) (23 more)) (23 than)) (23 two)) (23 months)))))))(3 .))
[S4]		(1 (20 (20 (17 (17 (17 (17 (23 For) (23 plated)) (23 through)) (23 holes)) (23 only))(2 :)) (13 (14 (11 (6 When) (10 (9 (23 the) (23 repair)) (8 (23 is) (23 completed))))(3 ,)) (10 (9 (9 (23 a) (23 clinched)) (23 lead-through)) (8 (8 (8 (8 (8 (8 (8 (23 is) (23 to)) (23 be)) (23 inserted)) (23 in)) (23 the)) (23 plated)) (23 through)) (23 hole)))))(3 .))

structure, allowing the annotation process to be more efficient. In order to subsequently convert the annotations into a format usable for training the RNTN, we implement a post-processing step. Specifically, we rebuild the binary structure for each annotated sentence. To this end, we implement an exporter, which transforms the annotations into a binary structured output. This can be illustrated by [S2] in Table 7.2. The four tokens *"is always 1290 bits"* are marked as a single Condition segment. However, an RNTN would expect three condition labels in this case, i.e. for the adjacent pairs [is, always], [is always, 1290] and [is always 1290, bits] if we apply left-branching. Our exporter sets these labels automatically and marks segments with brackets. The segment labels are represented by numbers to minimize the length of the binary annotations. Thus, the exporter creates the following binary annotation for the expression *is always 1290 bits*:

(8 (8 (8 (23 is) (23 always)) (23 1280))(23 bits))

Condition segments are tagged as 8, while Word segments are marked as 23. The exact functionality of our exporter[3] is defined by Algorithm 3.

7.2.2.5 Annotation Validity

In order to verify the reliability of the manual annotations, we calculated the inter-annotator agreement. For this purpose, we distributed the 1,571 conditionals among the four annotators, ensuring that 314 sentences are labeled by two annotators (overlapping quote of ≈20 %). Similar to other studies [302] that also utilize *brat* to annotate text segments, we calculate the pair-wise averaged F_1 score [303] based on the overlapping sentences. Specifically, we treat one rater as the subject and the other rater's answers as if they were a gold standard. This allows us to calculate the Precision and Recall values for their annotations. We then determine the F_1 score as the harmonic mean of Recall and Precision and take the average of F_1 scores among all pairs of raters in order to quantify the agreement of our raters: The higher the average F_1 score, the more the raters agree with each other.

For most of our manually assigned labels, we obtained an inter-annotator agreement of at least 0.83. The lowest agreement was achieved for Condition and And segments (0.73). The annotators did not always agree on how granular some expressions should be labeled (e.g., are there multiple conditions specified that need to be labeled as separate segments or can they be interpreted as one single segment?). The highest agreement was measured for the assignment of Statement segments (0.89). Based on the achieved inter-annotator agreement values, we assess our labeled data set as reliable and suitable for the implementation of our conditional extraction approach.

7.2.2.6 Data Analysis

Our final binary tree structured data set contains a total of 73,221 segments. Figure 7.5 provides an overview of the distribution of the segments across the individual labels. The distribution of segments is strongly unbalanced, since some segments (e.g., Conditional Statement) only occur on the upper levels of binary trees. Most of the segments represent Words, Conditions and Variables, because these labels are already

3 We share the code of the exporter in our *Github* repository: `https://github.com/springto/Fine-Grained-Causality-Extraction-From-NL-Requirements`

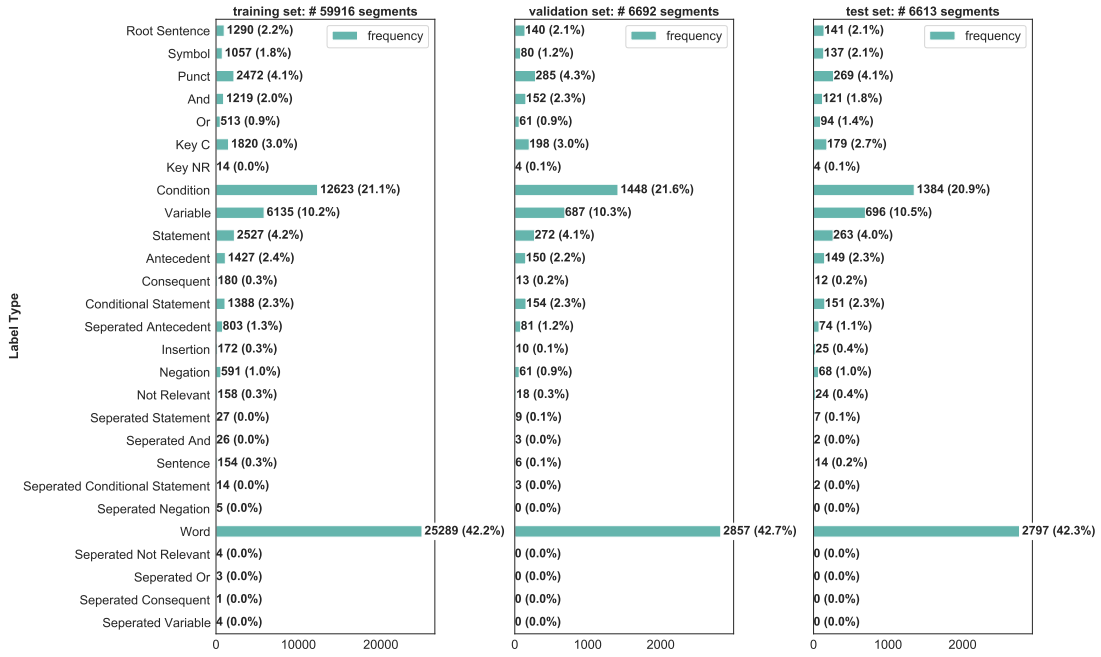


Figure 7.5: Overview of the Segment Distribution in Our Training, Validation, and Testing Data Sets.

assigned at the lower levels of the tree when multiple smaller text fragments are merged. A Root Sentence segment, on the other hand, occurs only once in a sentence. Hence, we find only 1,570 segments with this label in our data set. As shown in Figure 7.5, there are significantly more Antecedent segments than Consequent segments. One would expect that each conditional contains at least one *consequent*. Consequently, the number of *consequent* segments should be at least equal to the number of conditional statements. In the formulation of conditional statements, however, *consequents* are rarely explicitly introduced by cue phrases (e.g., *"then"*). In general, *antecedents* are introduced by cue phrases while the *consequents* are implicitly expressed by the semantics of the sentence. This often results in the combination of several Antecedent segments and Statement segments, which implicitly express the *consequent* (see S1, S3, and S4 in Table 7.2). Our first experiments have shown that it is important to distinguish between the explicit and implicit form of *consequents* during the annotation process, because otherwise the RNTN gets confused while learning the tree structures in bottom-up fashion. Thus, we annotate Statement segments only as Consequent segments if explicitly indicated by a cue phrase (see Step 4 in Section 7.2.2.3).

7.2.3 Implementation

This section presents the training and evaluation of the RNTN based on the *Conditional Treebank*. To determine the optimal configuration of the RNTN, we perform the following steps: First, we tune its hyperparameters. Second, we run two experiments and investigate whether we can improve the performance of the RNTN by using e.g. word vectors enriched with syntactic information.

Algorithm 3: Brat Annotation Export Algorithm

Data: Brat annotation file
Result: Binary tree structured sentence

```
initialize dataframes;
while annotations not empty do
    foreach sentence do
        convert annotations to dataframe containing begin and end positions of labels;
        append additional WORDCOUNT column to the data frame capturing the number
         of individual words in a label;
        create label containing final clause;
        add label to dataframe;
        if label spanning from beginning of sentence to predecessor of final clause exists then
            continue;
        else
            create label spanning from beginning to predecessor of final clause;
            add label to dataframe;
        foreach word in sentence do
            if word has single label then
                add label to dataframe;
            else
                create label WORD spanning from begin to end of word;
                add label to dataframe;
        foreach label in dataframe do
            if label has more than two childs then
                if label contains separator then
                    perform separator merge;
                else
                    if leftBranching then
                        for child = 1 to childsOfLabel do
                            if child+1 == childsOfLabel then
                                break;
                            else
                                create label from child to child+1;
                                add label to dataframe;
                    if rightBranching then
                        for child = childsOfLabel to 1 do
                            if child-1 == childsOfLabel then
                                break;
                            else
                                create label from child to child-1;
                                add label to dataframe;
```

7.2.3.1 Evaluation Procedure

We follow the idea of *Cross Validation* and divide the data set (1,571 sentences) in a training (1,290), validation (140) and test (141) set. Each segment is equally represented across all three data sets which helps to avoid bias in the prediction (see Figure 7.5). For example, in all three data sets, Condition segments represent about 20 % of all included segments. The training set is used to fit the algorithm while the validation set is used to tune its parameters. The test set is utilized for the evaluation of the algorithm based on real-world unseen data. We opt for a 10-fold *Cross Validation* as several studies have shown that a model that has been trained this way demonstrates low bias and variance [300]. We use standard metrics, for evaluating our approaches: Accuracy, Precision, Recall, and F_1 score. During the training process, we check the validation accuracy periodically in order to keep the model's checkpoint with the best validation performance.

7.2.3.2 Hyperparameter Tuning

We train the RNTN for 90 epochs seeking the optimal hyperparameter configuration. Specifically, we use *AdaGrad* as optimizer and set the learning rates (lr) to 0.1, 0.01, 0.001, and 0.0001. In addition, we try different mini batch (mb) sizes: 16, 24, 32, and 64. We set epsilon to 1e-08. As described in Section 2.3.2, each word needs to be represented as a d-dimensional vector. We try different dimension (wvecDim) sizes: 30, 50, and 60. Similarly to Socher et al. [129], we initialize all word vectors by randomly sampling each value from a uniform distribution: $\mathcal{U}$(-r, r), where r = 0.0001. Consequently, the word vectors are random at the beginning of the training process. However, we consider the word vectors as parameters that are trained jointly with the other parameters of the RNTN. We achieve the best performance with the following configuration: lr = 0.001, mb = 24 and wvecDim = 60. The model yields a training accuracy of 0.931 and a validation accuracy of 0.913 in epoch 87.

7.2.3.3 Setup of the Experiments

To further improve the performance of the RNTN, we conducted two experiments based on the identified optimal hyperparameter configuration.

POS Tagging Experiment Studies have shown that the performance of NLP models can be improved by providing explicit prior knowledge of syntactic information to the model [298, 4]. In this experiment, we investigate whether syntactic information also has a positive impact on the performance of the RNTN. We study two scenarios: First, the word vectors are randomly initialized and used as trainable parameters (as described in Section 7.2.3.2). Second, the word vectors are not randomly initialized but rather pre-trained and enriched with POS tags. Specifically, we add the corresponding POS tag to each token and create two vector representations: one for the actual token and one for the associated POS tag. We use the *nltk* library [304] to assign the POS tags to the respective tokens and *fastText* [143] to generate the pre-trained vectors.

As found during the hyperparameter tuning, the RNTN performs well with a vector dimension of 60. We stick to this dimension size and simply concatenate the pre-trained vector and the POS tag vector to a single representation. To investigate the impact

of the added syntactic information on the model performance, we concatenate the two vectors in three different ways. In the first variant, both vectors are equally weighted. The concatenated vector thus contains 30 dimensions representing the POS tag part and 30 dimensions for the pre-trained part (see Equation 7.5). In the second variant, we weight the syntactic information slightly more, so that the majority of the dimensions constitute the POS tag part (see Equation 7.6). Specifically, 75 % of the dimensions represent the POS tag part. In the third variant, the concatenated vector consists only of the POS tag part, i.e. the RNTN predicts only on the basis of POS tags and does not consider the actual token (see Equation 7.7). To measure the performance of the RNTN, we compare the test accuracy values achieved by using the different word vectors (see Figure 7.6a). The equations below illustrate the structure of the three word vector types. The POS tag part is highlighted in light green, while the pre-trained part is marked in dark green.

$$\begin{pmatrix} x_0 \\ \vdots \\ x_{29} \\ x_{30} \\ \vdots \\ x_{59} \end{pmatrix} \quad (7.5) \qquad \begin{pmatrix} x_0 \\ \vdots \\ \vdots \\ x_{44} \\ \vdots \\ x_{59} \end{pmatrix} \quad (7.6) \qquad \begin{pmatrix} x_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ x_{59} \end{pmatrix} \quad (7.7)$$

Branching Experiment In this experiment, we study whether the branching method used to generate the binary tree-structured data influences the performance of the RNTN. Specifically, we create two data sets with our tree exporter: one data set in which the adjacent pairs are merged by left branching and one data set in which we apply right branching. We also build a third data set that combines the left and right branching data in order to further increase the number of training instances. We divide each of the three data sets into a training, validation, and test set and train the RNTN on these sets. To measure the performance of the RNTN, we compare the respective test accuracy values (see Figure 7.6b). We build on the findings of the POS tagging experiment and use the word vectors with a weighting of 50:50, as this results in the best performance of the model.

7.2.3.4 Results of the Experiments

Figure 7.6 reports the results of our experiments. In this section, we interpret the results and select the best-performing model as our final conditional extractor.

POS Tagging Experiment Irrespective of the selected word vectors, the RNTN achieves a promising result of at least 84 % test accuracy over all n-grams lengths (see Figure 7.6a). We achieve the best performance by using word vectors with the POS tag and pre-trained parts weighted equally. A comparison of the performance between the three different POS tag weights shows that the higher the proportion of POS tags in the word vector, the lower the test accuracy. In fact, the model achieves a test accuracy of 91.2 % with the 50:50 weighting, 86.2 % with the 75:25 weighting, and the lowest value of 84.1 % with the 100:0 weighting. We hypothesize that only POS tags are not sufficient to comprehend a conditional since they only reflect the syntax of a sentence, but not its semantics.

Therefore, the model performs better if it considers both the POS tag and the actual token during prediction.

Interestingly, even if the word vectors are randomly initialized and treated as trainable parameters, the RNTN shows a very good test accuracy of 90.4 %. The tuned word vectors outperform the word vectors with a 75:25 weighting by 4.2 % and the word vectors with a 100:0 POS tag weighting by 6.3 %. For segments with a short length of up to 5-grams, the trainable vectors even outperform the 50:50 weighted vectors. With increasing n-gram length, however, the RNTN shows better performance when using the syntactically enriched vectors. Over all test instances, the difference between the two test accuracy values is small (only 0.8 %). Consequently, only a marginal performance gain could be achieved by adding syntactic information to the word vectors.

Branching Experiment Figure 7.6b shows the performance of the RNTN depending on the selected branching method. Similar to the previous experiment, the RNTN performs well on all three test sets and achieves at least 85 % test accuracy. Interestingly, the RNTN achieves a better test accuracy when applying left branching rather than right branching (difference of 6.2 %). Combining the left and right branched data sets, the RNTN achieves a test accuracy of 88 %. Hence, our experiment demonstrates that the RNTN is better at building the binary parse tree using left-branching than right-branching.

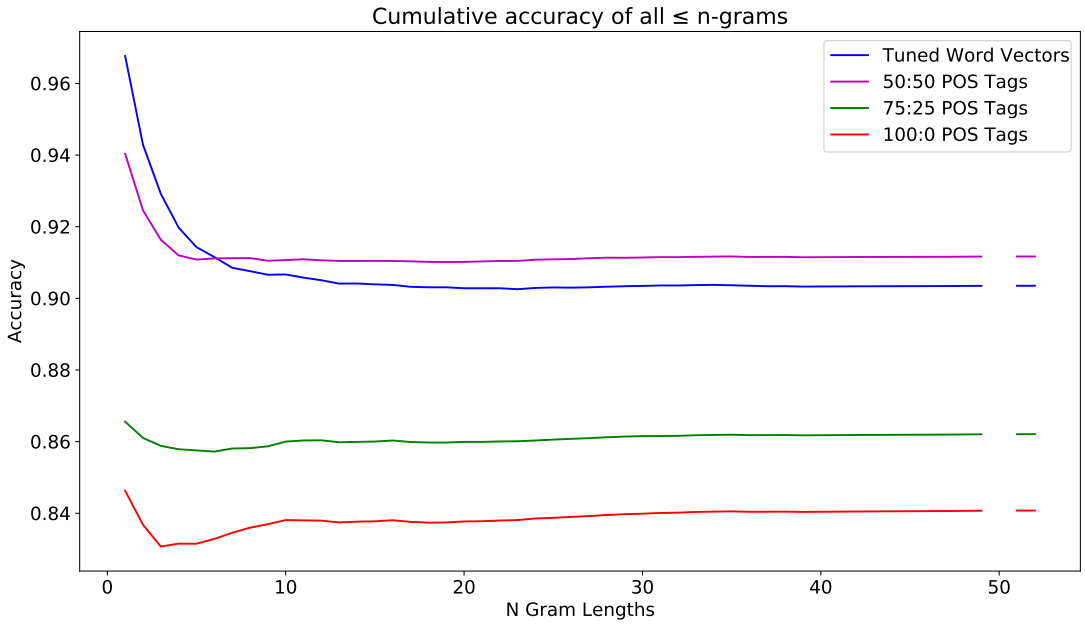
Summary of Experiments:

We found that enriching word vectors with POS tags does not necessarily lead to a significant performance gain. If POS tags account for more than half of the dimensions of the vector, the RNTN performs worse compared to when the vectors are randomly initialized and trained jointly with the model. The RNTN seems to learn left branching better than right branching.

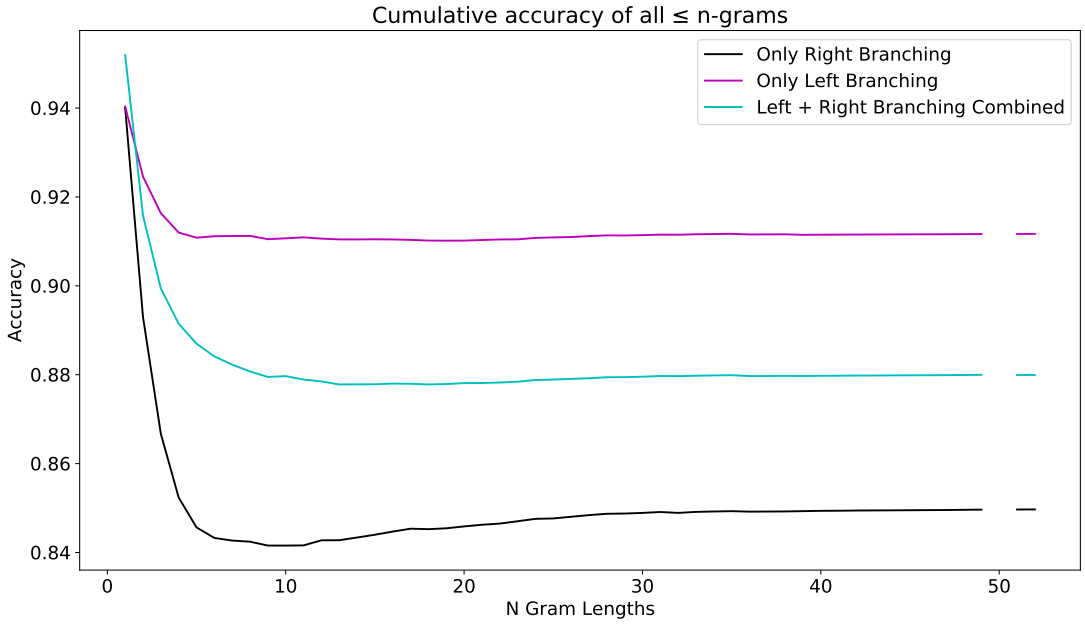
7.2.4 Evaluation

The test accuracy values achieved in our experiments already indicate that the RNTN is able to parse conditional statements. However, we are not only interested in the overall test accuracy, but also in the performance with respect to the individual segments. Table 7.3 presents the Recall, Precision and F_1 scores per segment.

Potentials We observe that the RNTN predicts a number of segments reliably. It achieves a F_1 score of at least 90 % for the segments Root Sentence, Punct, Variable, Statement, Antecedent, Conditional Statement, Separated Antecedent, Insertion, Negation and Word. Across all segments, our approach yields a F_1 score of 74 %. Not surprisingly, the RNTN is able to predict segments like Root Sentence, Punct and Word almost perfectly, since these segments always occur on the same level of the tree: the Root Sentence segment corresponds to the root of the tree and can thus be found on top, while the segments Punct and Word do always occur at the bottom of the tree. In addition, Punct and Word segments always represent 1-grams. Contrary, the RNTN shows a poor performance for the segments Key NR, Not-relevant, Sentence and Separated Conditional Statement. We hypothesize that the poor performance stems



(a) POS Tagging Experiment.



(b) Branching Experiment.

Figure 7.6: Accuracy Curves for Fine-Grained Conditional Extraction at Each N-Gram Lengths. Both Plots Show the Cumulative Accuracy of All ≤ n-grams. Upper Figure: Impact of Tuned Word Vectors and Syntactically Enriched Word Vectors on Performance. Lower Figure: Impact of Branching Methods on Performance.

Table 7.3: Results of the Evaluation on the *Conditional Treebank*. The Segments Separated Negation, Separated Not-relevant, Separated Or, Separated Consequent, and Separated Variable are Not Included in the Test Set and Are Therefore Not Part of the Evaluation. F_1 Scores of at Least 0.9 Are Marked in **Bold**.

	Performance Measures		
Label Type	Recall	Precision	F_1 - Score
Root Sentence	1.0	1.0	**1.0**
Symbol	0.2	0.58	0.39
Punct	1.0	0.93	**0.97**
And	0.82	0.65	0.74
Or	0.79	0.94	0.87
Key-C	0.86	0.87	0.87
Key-NR	0.0	0.0	0.0
Condition	0.82	0.78	0.8
Variable	0.88	0.91	**0.9**
Statement	0.93	0.9	**0.92**
Antecedent	0.95	0.95	**0.95**
Consequent	0.83	0.83	0.83
Conditional Statement	0.93	0.94	**0.94**
Separated Antecedent	0.98	0.96	**0.97**
Insertion	0.88	0.95	**0.92**
Negation	0.94	0.92	**0.93**
Not-relevant	0.0	0.0	0.0
Separated Statement	0.42	1.0	0.71
Separated And	1.0	0.66	0.83
Sentence	0.21	0.6	0.41
Separated Conditional Statement	0.5	0.33	0.42
Separated Negation	-	-	-
Word	0.99	0.93	**0.96**
Separated Not-relevant	-	-	-
Separated Or	-	-	-
Separated Consequent	-	-	-
Separated Variable	-	-	-
Mean	0.72	0.76	0.74

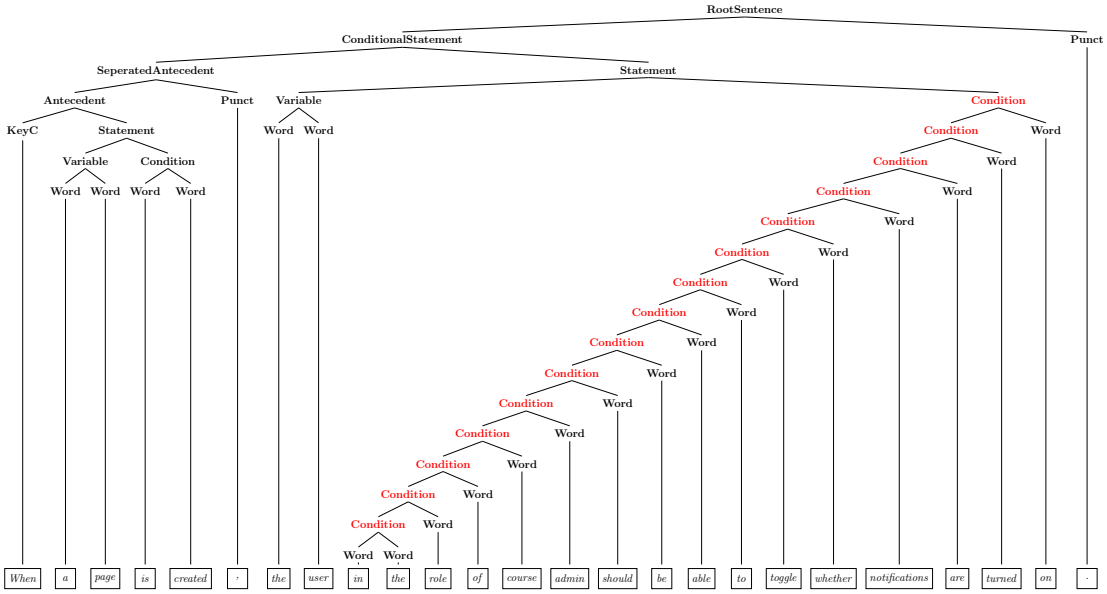
from the fact that these segments are highly under-represented in the training and validation set (see Figure 7.5), rendering them difficult for the RNTN to learn. The RNTN seems to detect well which tokens in a NL sentence represent an *antecedent*. However, it shows a weaker performance in predicting *consequents* (12 % difference in both Recall and Precision). We hypothesize that this results from the significant under-representation of the explicit form of *consequents* in contrast to its implicit form (see Figure 7.5). The strong performance with respect to the Conditional Statement segment (F_1 score of 94 %) shows that the RNTN acquired the concept of conditionals being a combination of single/multiple Antecedent and Consequent segments. Interestingly, the RNTN achieves a better F_1 score for the prediction of Or segments than for And segments. Predicting And segments seems to be a more difficult task than the prediction of Or segments, because the latter usually contain an *"or"* token, while And segments often contain several Condition segments which are concatenated without an *"and"* token (see [S3] in Table 7.2). In these cases, the conjunction is implicitly contained in the semantics of the sentence and not by an explicit *"and"* token, making the prediction challenging.

Limitations When analyzing the test predictions, we found that the RNTN sometimes fails to distinguish between Variable and Condition segments, i.e. it assigns tokens that actually belong to a Variable segment to a Condition segment and vice versa. Across all test predictions, we observed that the RNTN has a slight bias towards Condition segments and tends to construct large separate Condition segments. This leads to a significant number of *False Positives* (Precision value of only 78 %) and can even result in a complete false parse tree as indicated by Figure 7.7a.

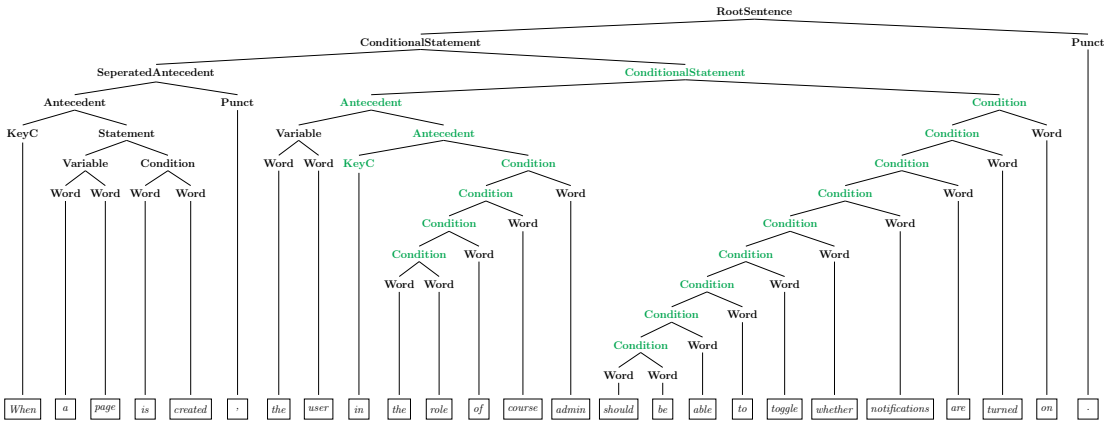
In this example, the RNTN detects only the outer *antecedent* (*"a page is created"*) and ignores that the sentence contains a second *antecedent*: *"only users with admin rights are allowed to view the notification settings."* Rather, the RNTN constructs a large distinct Condition segment and merges it with the Variable segment [the user]. As a result, the binary tree is assembled incorrectly, because the inner conditional is not recognized. This example illustrates one of the major limitations of the RNTN. Due to the bottom-up construction of the tree, prediction errors on the lower layers strongly affect the upper layers. Initial experiments revealed that this constitutes a problem especially when we apply the RNTN to words that are not yet part of its training's vocabulary. In such cases, the RNTN is unsure already on the lower layers to which segments the unknown tokens should be assigned and struggles to understand the semantics of the sentence. These errors are propagated to the upper layers meaning that the RNTN builds the binary parse tree based on wrong segments.

Summary of Evaluation:

The RNTN predicts most of the segments with high Precision and Recall. It shows a better performance in detecting *antecedents* than explicit *consequents*. We see potential for optimization in the detection of conjunctions as well as in the splitting of *antecedents* and *consequents* into variable and condition. A threat to the practicability of our approach is that early prediction errors have a major negative impact on the composition of the upper layers.



(a) Incorrect Binary Parse of the RNTN. Improperly Identified Segments Are Marked in Red.



(b) Correct Binary Parse. Segments That the RNTN Did Not Detect Are Highlighted in Green.

Figure 7.7: Overview of Binary Parse Trees Representing the Sentence: *"When a page is created, the user in the role of course admin should be able to toggle whether notifications are turned on."* Upper Figure: Prediction of Our Trained RNTN. Lower Figure: Semantically Correct Binary Parse Tree.

7.3 Extracting Conditionals by Sequence Labeling

This section introduces our third approach for extracting conditional statements from natural language. In Section 7.3.1, we describe the principal idea behind using multi-class and multi-label models for conditional extraction, and explain the key differences compared to our first and second approach. Further, we detail the creation of an appropriate corpus suitable for training our approach (see Section 7.3.2). Section 7.3.3 outlines the functionality and implementation of our conditional approach based on multi-class and multi-label models. We evaluate our approach in Section 7.3.4.

7.3.1 Principal Idea

We define the extraction of conditionals as a sequence labeling problem, in which we are given a certain NL sentence $\mathcal{X}$ in the form of a sequence of n tokens $\mathcal{X} = \{x_i\}_{i=1}^{n}$ and we are required to produce a sequence $\mathcal{Y}$ of corresponding token labels. Specifically, we aim to demarcate tokens that are part of a conditional from tokens that should be excluded from further processing. In our case, we are interested in twelve token labels. Since conditionals in requirements usually consist of up to three *antecedents* and *consequents* (see Chapter 4), we create individual labels for each *antecedent* and *consequent* to clearly separate them.

1. Antecedent 1
2. Antecedent 2
3. Antecedent 3

} Antecedent Labels

4. Consequent 1
5. Consequent 2
6. Consequent 3

} Consequent Labels

7. Not Relevant: Marks parts of a sentence that are not part of the conditional.
8. And: Marks a conjunctive link between two adjacent *antecedents* or *consequents*.
9. Or: Marks a disjunctive link between two adjacent *antecedents* or *consequents*.
10. Variable: Marks the variable of an *antecedent* or *consequent*.
11. Condition: Marks the condition of an *antecedent* or *consequent*.
12. Negation: Marks negated *antecedents* or *consequents*.

We use these token labels to generate two annotation layers (see Figure 10.1). The top layer represents the composition of the sentence by specifying the *antecedents*, *consequents*, and their combinatorics based on the labels 1 - 9. At the lower layer, we use the labels 10 - 12 to annotate the *antecedents* and *consequents* more fine-grained. Consequently, we assign at least one label and at most two labels to a token. Our sequence labeling problem can be solved in two ways: One way is to consider the creation of both annotation layers as two separate multi-class classification tasks. Accordingly, we train two models,

where the first model is responsible for recognizing *antecedents* and *consequents*, while the second model splits the *antecedents* and *consequents* into variables and conditions. In this case, the first model produces the top layer and the second model creates the lower layer. Alternatively, we treat our annotation task as a multi-label classification problem. Consequently, we train only one model, which considers all labels during the prediction and assigns multiple labels to a token. We conduct experiments with both multi-label and multi-class classifiers and select the best approach for the fine-grained conditional extraction. Utilizing multi-label and multi-class classifiers based on language models such as BERT, RoBERTa, and DistilBERT for conditional extraction has several advantages compared to our approaches presented in Section 7.1 and Section 7.2:

- **Increased Robustness Against Unknown Words:** Modern language models such as BERT decompose each sentence using subword tokenization algorithms (see Section 2.3.3). Specifically, common words are not split into smaller subwords, while rare words are decomposed into meaningful subword units that are known to the model. Consequently, models such as BERT are able to process *Out-Of-Vocabulary* words as they can always create tokens for a given sequence, independent of whether they have seen each word before.
- **Decreased Propagation of Early Prediction Errors:** Our *Dependency Parsing*-based approach and RNTN-based approach aim at parsing conditionals by predicting their composition as a tree-structure. As our evaluation shows, the prediction of a tree structure suffers from the limitation that prediction errors on the lower layers negatively affect the prediction of the higher levels. Our sequence labeling approach predicts at the token level and does not consider previously made predictions when assigning labels to a token. Instead, we leverage the benefit of BERT's self attention mechanism and assign labels based on the token embeddings that have been contextualized with information from the entire input sequence. Thus, *early* prediction errors do not affect the overall performance of our approach.

7.3.2 Training Corpus Creation

To train our third conditional extraction approach, we require an annotated data set in which the combinatorics of *antecedents* and *consequents* as well as their variables and conditions are labeled. We can not reuse the corpus that we have already used for training our RNTN-based approach, because it contains annotated binary trees that are not suitable to train a sequence labeling method (see Section 7.2). Other existing data sets [305] are also not suitable for our use case: The *SemEval-2007* [306] and *SemEval-2010* [216] data sets contain only single word causal pairs. In the data set presented by Dasgupta et al. [70], *antecedents* and *consequents* are only coarsely annotated (i.e., connectives, variables, and conditions are not labeled). Due to the unavailability of adequate data, we create our own training corpus. To this end, we build on the data set that we have already used for training our detection algorithm. We randomly select a subset of the requirements that contain conditionals (1,946) and annotate them using our twelve predefined labels.

Annotation Process We involve four annotators with previous experience in the interpretation of conditionals and conduct a workshop where we discuss several examples.

Table 7.4: Overview of the Class Distribution (Sentence and Token Level) in Our Training, Validation and Testing Data Sets. Annotation Validity per Class Is Reported as Pair-Wise Averaged F_1 Score.

		Complete Dataset			Training Set			Validation Set			Testing Set			
	Label Type	Sent.	WordPiece Tokens	BPE Tokens	Sent.	WordPiece Tokens	BPE Tokens	Sent.	WordPiece Tokens	BPE Tokens	Sent.	WordPiece Tokens	BPE Tokens	Annotation Validity
Top Layer	**Antecedent 1**	1946	18743	18317	1556	14862	14499	194	1878	1848	196	2003	1970	87 %
	Antecedent 2	661	5158	5190	523	4036	4072	68	540	534	70	582	584	71 %
	Antecedent 3	137	1109	1102	105	856	853	18	123	117	14	130	132	71 %
	Consequent 1	1946	22814	22115	1556	18370	17832	194	2210	2125	196	2234	2158	90 %
	Consequent 2	614	5384	5426	483	4169	4200	65	573	578	66	642	648	81 %
	Consequent 3	138	1129	1142	113	952	958	13	80	82	12	97	102	78 %
	Not Relevant	664	6667	6371	537	5407	5175	60	631	595	67	629	601	74 %
	And	744	2799	2807	590	2215	2221	80	297	298	74	287	288	93 %
	Or	230	826	826	182	667	667	26	87	87	22	72	72	91 %
Lower Layer	**Variable**	1946	26076	25753	1556	20896	20599	194	2543	2509	196	2637	2645	87 %
	Condition	1946	34927	34974	1556	27653	27738	194	3513	3498	196	3761	3738	81 %
	Negation	363	1458	1513	287	1154	1199	34	133	139	42	171	175	90 %

To ensure consistent annotations, we create an annotation guideline, in which we define each label along with a set of sample annotations. We use the web-based *brat* annotation platform [301] for labeling each sentence.

Annotation Validity To verify the reliability of the annotations, we calculate the inter-annotator agreement. We distribute the 1,946 sentences containing conditionals among four annotators, ensuring that 390 sentences are labeled by two annotators (overlapping quote of ≈20 %). Similar to other studies [302] that also utilize *brat* to annotate sentences, we calculate the pair-wise averaged F_1 score [303] based on the overlapping sentences. Specifically, we treat one rater as the subject and the other rater's answers as if they were a gold standard. This allows us to calculate Precision and Recall for their annotations. We then determine the F_1 score as the harmonic mean of Recall and Precision. We calculate the F_1 score pairwise between all raters. Subsequently, we take the average of F_1 scores among all pairs of raters to quantify the agreement of our raters: The higher the average F_1 score, the more the raters agree with each other.

For most of our labels, we obtain an inter-annotator agreement of at least 81 % (see Table 7.4). The lowest agreement is achieved for Antecedent 2 and Antecedent 3 (F_1 score of 71 %). The annotators do not always agree on how granular some expressions should be labeled (e.g., does a text fragment represent another *antecedent*, or is it still part of the previous *antecedent*?). The highest agreement is measured for the assignment of And labels (F_1 score of 93 %). Averaged across all labels, we achieve an F_1 score of 83 %. Based on the achieved inter-annotator agreement values, we assess our labeled data set as reliable and suitable for the implementation of our conditional extraction approach.

Data Analysis Table 7.4 shows that the majority of our sentences contain only a single *antecedent* and *consequent*. About one third of the sentences contain more complex conditionals comprising two *antecedents* or two *consequents*. Only a few sentences contain three *antecedents* or three *consequents*. We found that *antecedents* and *consequents* are more often connected by a conjunction than by a disjunction. Negated *antecedents* and *consequents* occur in about 18 % of the sentences. At the token level, expressions labeled as Consequent 1 are often longer than Antecedent 1 expressions. We observe a similar trend at the lower layer. Conditions are usually longer than the Variables of the *antecedents* and *consequents*. Across all classes, our data set is strongly unbalanced with four minority classes: Antecedent 3, Consequent 3, Or, and Negations.

7.3.3 Implementation

We implement and compare the performance of nine different models for multi-class and multi-label classification. Table 7.5 provides an overview of their architectures consisting of three different layers: embedding layer, *Bidirectional LSTM* (BiLSTM) layer, and inference layer.

Layer 1: Embedding Layer Similar to the detection of conditionals, we also use contextual word embeddings for their extraction (see Chapter 6). However, we do not only perform experiments with BERT but also investigate the influence of RoBERTa [135] and DistilBERT [136] embeddings on the performance of our models. To extract the conditionals in fine-grained form, we need to predict on the token level. Hence, in contrast to

Table 7.5: Overview of Architecture and Evaluation Results of All Trained Models. Best *macro*-F_1 Score Is Marked in **Bold**. Tuned Hyperparameters Are Reported in Terms of Batch Size (bs), Learning Rate (lr), Dropout (d), and Size of Hidden State.

		Architecture				
	</> model #	Embedding	BiLSTM	Inference	*Macro*-F_1	Optimal Hyperparameters
2x Multi-class Models	</> model I	BERT	✘	Softmax	76 %	top layer model: bs= 32, lr = 7.49e-05, d = 0.27 lower layer model: bs = 64, lr = 6.24e-05, d = 0.18
	</> model II	RoBERTa	✘	Softmax	75 %	top layer model: bs = 32, lr = 6.24e-05, d = 0.33 lower layer model: bs = 32, lr = 4.28e-05, d = 0.21
	</> model III	DistilBERT	✘	Softmax	78 %	top layer model: bs = 32, lr = 8.80e-05, d = 0.32 lower layer model: bs = 64, lr = 9.76e-05, d = 0.36
1x Multi-label Model	</> model IV	BERT	✘	Sigmoid	85 %	bs = 64, lr = 8.79e-05, d = 0.26
	</> model V	RoBERTa	✘	Sigmoid	**86 %**	bs = 32, lr = 6.13e-05, d = 0.13
	</> model VI	DistilBERT	✘	Sigmoid	82 %	bs = 32, lr = 4.47e-05, d = 0.14
	</> model VII	BERT	✓	Sigmoid	83 %	bs = 32, lr = 6.27e-05, lstm_hidden = 128, d= 0.31
	</> model VIII	RoBERTa	✓	Sigmoid	84 %	bs = 32, lr = 4.34e-05, lstm_hidden = 128, d = 0.01
	</> model IX	DistilBERT	✓	Sigmoid	72 %	bs = 32, lr = 9.437e-05, lstm_hidden = 128, d = 0.50

the detection algorithm where we consider only the CLS token during classification, we pass each token to the classifier, which assigns a label to each token. We consider both actual tokens of a sentence and synthetically added tokens (PAD, SEP, and CLS), because initial experiments demonstrated that the exclusion of synthetic tokens results in significant performance degradation (loss of ≈5% in *macro*-F_1).

The number of tokens per class differs depending on the applied tokenizer (see Table 7.4). BERT and DistilBERT use *WordPiece* as a subword tokenization algorithm, while RoBERTa employs *Byte-Pair Encoding* (BPE). Nevertheless, both tokenizers differ only slightly, so that we set the same maximum length of tokens per sentence for all models. By analyzing the annotated conditionals, a maximum length of 80 tokens proved to be reasonable.

Layer 2: Bidirectional-LSTM Layer For </> models VII - IX, we feed the word vectors into a BiLSTM to obtain a hidden state for each word. *Bidirectional Long Short-Term Memory* models have demonstrated to be well suited for sequence labeling problems, because they consider both the past and future contexts of the words. To enable the hidden states to capture both historical and future context information, we train two LSTMs on the input sequence. The forward LSTM processes the sentence from v_1 to v_n, while a backward LSTM processes from v_n to v_1. Consequently, we obtain two hidden states at each time step t. $\overrightarrow{h}$ is computed based on the previous hidden state $\overrightarrow{h}_{t-1}$ and the input at the current step v_t, while $\overleftarrow{h}$ is computed based on the future hidden state $\overleftarrow{h}_{t+1}$ and

the input at the current step v_t. We obtain the final hidden state by concatenating the forward and backward context representations:

$$h_i = \overrightarrow{h_i} \oplus \overleftarrow{h_i} \tag{7.8}$$

Layer 3: Inference Layer For </> models I - III, we put the word vectors into a single-layer feedforward neural network that outputs the final predicted tag sequence for the input sentence. We use a softmax layer, which calculates the class probabilities for each token:

$$\hat{y} = \text{softmax}(Wv_i + b) \tag{7.9}$$

We define $\hat{y}$ as the predicted label probabilities for the i-th token, W as the weighted matrix, and b as the bias. We select the class with the highest probability as the final classification result. Since we train two different models for the annotation of the top and lower layer, we apply two different softmax functions. The model predicting the top layer considers nine labels (see Equation 7.10) while the lower layer is annotated with only three labels (see Equation 7.11):

$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{9} exp(x_j)} \tag{7.10}$$

$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{3} exp(x_j)} \tag{7.11}$$

In case of the </> models IV - VI, we use a sigmoid layer to perform multi-label classification:

$$\hat{y} = \text{sigmoid}(Wv_i + b) \tag{7.12}$$

We select the classes with a probability ≥ 0.5 as the final classification result. In case of the </> models VII - IX, we consider the hidden states as the feature vectors. Consequently, we define the sigmoid layer as:

$$\hat{y} = \text{sigmoid}(Wh_i + b) \tag{7.13}$$

7.3.4 Evaluation

This section reports on the results of our experiments, in which we compare the performance of the individual models.

7.3.4.1 Evaluation Procedure

We divide the data set (1,946 sentences) into a training (1,556), validation (194), and test (196) set. Each class is equally represented across all three data sets, which helps to avoid bias in the prediction (see Table 7.4). We use Precision, Recall, and F_1 score for evaluating our models. Since our data set is strongly unbalanced, we need to interpret the metrics carefully. In particular, it is important to distinguish between *macro* and *micro* averages of the metrics. *Macro*-averaging involves the computation of the metrics per class and then averaging them. Hence, each class is treated equally. *Micro*-averaging combines the contributions of all classes to calculate the mean. Thus, it takes label imbalance into account and favors majority classes. In our use case, all classes are equally important. Predicating a minority class like Or is as crucial as predicting a majority

Table 7.6: Performance of </> model V per Individual Label. *Macro*-F_1 Scores of at Least 90 % Are Marked in **Bold**.

	Label Type	Precision	Recall	Macro F_1
Top Layer	**Antecedent 1**	92 %	89 %	**91 %**
	Antecedent 2	83 %	72 %	77 %
	Antecedent 3	76 %	88 %	82 %
	Consequent 1	90 %	89 %	**90 %**
	Consequent 2	83 %	85 %	84 %
	Consequent 3	57 %	76 %	65 %
	Not Relevant	91 %	92 %	**91 %**
	And	94 %	96 %	**95 %**
	Or	85 %	92 %	88 %
Lower Layer	**Variable**	87 %	92 %	89 %
	Condition	93 %	89 %	**91 %**
	Negation	79 %	90 %	84 %

class like Antecedent 1, because it has a major impact on capturing the combinatorics in a sentence. Therefore, we choose the *macro*-F_1 score as our main evaluation criterion.

7.3.4.2 Hyperparameter Tuning

The performance of DL models depends heavily on the network architecture as well as the hyperparameters used. Therefore, we compare the performance of our models using different hyperparameter configurations. To determine the optimal hyperparameters, we use the *Tree-structured Parzen Estimator* algorithm [307]. During the training process, we check the validation *macro*-F_1 score periodically to keep the model's checkpoint with the best validation performance. We train our models for 50 epochs on the training data with a patience of 5 epochs. Table 7.5 shows the best hyperparameters for each model.

7.3.4.3 Results

We first compare the overall performance of our trained models across all classes. In addition, we study the impact of the different word embeddings and the BiLSTM layer on the model performance. Finally, we investigate the performance of our best model to predict the individual labels.

Overall Comparison The achieved *macro*-F_1 scores demonstrate that all investigated models are able to extract conditional statements in fine-grained form (see Table 7.5). However, we observe significant performance gaps between the multi-class and multi-label models. On average, the multi-class models obtain a *macro*-F_1 score of 76.34 % while the multi-label models yield an average *macro*-F_1 score of 82 %. Consequently, the multi-label models seem to be more suitable for our use case. </> Model V demonstrates the best performance with a *macro*-F_1 score of 86 %, which represents a performance gain of 8%

compared to the best multi-class </> model III. We do not witness a major performance difference among most of the multi-label models. In fact, </> model IV shows a very similar behavior as </> model V and achieves a *macro-F_1* score of 85 %. </> Model IX, however, represents an outlier and produces the poorest *macro-F_1* score of all trained models.

Impact of Embeddings Our experiments reveal that the choice of embeddings has an impact on the prediction performance. The best performance of the multi-class models is achieved by using the DistilBERT embeddings (see Table 7.5). In contrast, the multi-label models show the best performance when building on RoBERTa, regardless of the usage of a BiLSTM. Interestingly, the selection of embeddings has the greatest impact on the models that use a BiLSTM for feature extraction. For example, a comparison of the performance of </> model VIII and </> model IX reveals a performance gap of 12 % in *macro-F_1*. In the case of the other models, the performance differences are considerably smaller: the performance of </> model II and </> model III differ by only 3 % in *macro-F_1*, while </> model V and </> model VI deviate by only 4 % in *macro-F_1*.

Impact of BiLSTM Layer In our setting, adding the BiLSTM layer did not lead to any performance improvement. In fact, the multi-label models demonstrate better performance without the BiLSTM. We hypothesize that our amount of training instances is not adequate to sufficiently train the complex BiLSTM architecture and take advantage of its benefits.

Label Prediction Table 7.6 indicates that </> model V is capable of processing both conditional statements consisting of only one cause as well as conditionals with multiple *antecedents*. Our model predicts Antecedent 1 with very high Precision and Recall resulting in a *macro-F_1* score of 91 %. For the prediction of Antecedent 2 and Antecedent 3, our model also performs well by achieving *macro-F_1* scores of 77 % and 82 %, respectively. Conditionals that contain only one *consequent* or two *consequents* can also be processed well by our model. However, our experiments show that the model lacks certainty in the prediction of Consequent 3 (*macro-F_1* score of only 65 %). We assume that this stems from its under-representation in our training set. The highest *macro-F_1* score is achieved by </> model V for the prediction of And. Likewise, our model performs well in recognizing tokens representing disjunctions (*macro-F_1* score of 88 %). This indicates that our model is able to understand and extract the combinatorics of *antecedents* and *consequents*. In addition, the model performs very well in detecting tokens that are not relevant for test case generation. Our experiments prove that our model performs well in predicting both the top and lower layers. The obtained *macro-F_1* scores for Variable and Condition show that our model is able to decompose *antecedents* and *consequents* into more granular fragments. In addition, </> model V reliably identifies negations within the conditionals.

Summary of Evaluation:

Our experiments reveal that </> model V is best suited to extract conditionals in fine grained form. Specifically, the combination of RoBERTa embeddings (embedding layer) and a sigmoid classifier (inference layer) achieved the best performance. We therefore use </> model V for the second step in the `CiRA` pipeline.

Chapter 8

Conclusions and CiRA Overview

Common Thread Our experiments presented in Chapter 6 and Chapter 7 demonstrate that the detection of conditionals in NL requirements and their extraction is a challenging task. In this chapter, we summarize the main findings of our experiments and present CiRA as a solution to the second problem addressed in this thesis, namely *"the fine-grained extraction of conditionals from RE artifacts"* (see Section 1.3).

Contribution CiRA solves conditional extraction as a two-step problem: It first detects whether requirements contain conditional statements. Second, if they contain conditionals, it interprets and extracts them in fine-grained form. CiRA builds on the findings of our experiments and employs the models found to be the most performant for each of the two steps. Hence, it uses syntactically enriched BERT embeddings combined with a softmax classifier for the detection of conditionals, and a sigmoid classifier built on RoBERTa embeddings to extract them in fine-grained form. This chapter condenses the functionality of CiRA and introduces a tool that allows fellow researchers and practitioners to easily interact with CiRA. CiRA is publicly available at `www.cira.bth.se/demo/`.

Related Publications This chapter is taken, directly or with minor modifications, from a previous publication [10].

8.1 The CiRA Pipeline

As shown in Figure 8.1, CiRA consists of two steps: It first detects whether an NL requirement contains a conditional. Second, it extracts the conditional in fine-grained form. Specifically, CiRA considers the combinatorics between *antecedents* and *consequents* and splits them into more granular text fragments (e.g., variable and condition), making the extracted conditionals suitable for automatic test case derivation. We have implemented and compared different methods for both steps and incorporated the best performing methods into the pipeline of CiRA. We describe the functionality of CiRA by means of the following requirement: *"If A is valid and B is false, then C is true."*

Step 1: Detection of Conditionals As shown in Section 6.3, enriching input sequences with dependency tags leads to a better performance of our conditional detection approach. We therefore, in the first step, use *spaCy* to assign dependency tags to the individual tokens in the sentence. This allows our conditional classifier to take into account not only the content of the tokens themselves, but also the grammatical structure of the sentence when categorizing a sentence into the two classes Conditional Present and Conditional Not Present. In case of our exemplary requirement, the token *"If"* is assigned the dependency tag *mark*, indicating that *"If"* introduces a clause subordinate to another clause. After allocating appropriate dependency tags to each token in the sentence, the sentence is decomposed using the *WordPiece* tokenizer and enriched with additional synthetic tokens such as the CLS token. Finally, we feed each token into the BERT model to generate word embeddings. Since we perform conditional detection at sentence level, we only pass the CLS token into the softmax classifier, which computes the probability of whether the input sequence contains a conditional or not. The classifier calculates a confidence of 91 % that our exemplary requirement contains a conditional. With only 9 % confidence, our classifier assumes that the input sequence does not contain a conditional. Our approach selects the category with the highest confidence and classifies our example correctly as Conditional Present. The detected conditional is passed to the next step of the pipeline.

Step 2: Fine-grained Extraction of Conditionals In the second step, we utilize the BPE tokenizer to convert the detected conditional statement into a form that can be processed by RoBERTa. After decomposing the input sequence into individual tokens, we pass each token into the RoBERTa model to create word embeddings. Since we perform conditional extraction at token level, we feed the embeddings of all tokens into our sigmoid classifier. Our classifier calculates the probability for each class whether a given token should be assigned to that class or not. Since we differentiate between twelve different classes (see Section 7.3.1), the sigmoid classifier calculates twelve probabilities accordingly. We select the classes with a probability ≥ 0.5 as the final classification result. In case of our exemplary requirement, the *"If"* token is classified as Not Relevant with a confidence of 99.6 %. The token *"A"* is assigned to two classes: On the bottom annotation layer, *"A"* is correctly marked as a Variable. On the top annotation layer, *"A"* is identified as belonging to Antecedent 1. The synthetically added tokens by the BPE tokenizer like *"<s>"* and *"<pad>"* are correctly identified as Not Relevant. We follow the classifications of our sigmoid classifier and assign the corresponding labels to each token of the input

sequence. The output of the CiRA pipeline thus represents a list of top layer and bottom layer labels, allowing us to annotate the conditional in fine-grained form.

8.2 Tool Support

To facilitate an easy interaction with CiRA, we implemented a corresponding tool support. We invite fellow researchers and interested practitioners to employ CiRA at `www.cira.bth.se/demo/`. Our tool does not only allow the use of CiRA for the fine-grained extraction of conditionals from NL sentences, but also enables the automatic derivation of acceptance tests based on the extracted conditionals. Specifically, our tool realizes Use Case 1 by combining CiRA with *Cause-Effect-Graphing* to create acceptance tests automatically. A detailed description of how we derive acceptance tests from the extracted conditionals is given in Chapter 10. In this section, we only present the corresponding tool support.

Technical Setup and User Interface Our website is built as a restful node.js server utilizing the *Express* framework. The backend's main purpose is to execute a *Python* script, which serves as a wrapper around our conditional classifier and conditional extraction algorithm. As illustrated by Figure 8.2, our tool-supported approach consists of four components: 1) *Detection of Conditionals*, 2) *Extraction of Conditionals*, 3) *Creation of Cause-Effect-Graph*, and 4) *Creation of Acceptance Test*. We outline all four components below and use the following requirement as our running example: *"If the temperature change is requested, then the determine heating/cooling mode process is activated and makes a heating/cooling request."*

Detection of Conditionals The UI provides a text input field, in which an arbitrary NL sentence can be entered (see Figure 8.2a). Upon pressing the *"classify"*-button, the sentence is sent to the backend where it is processed by the aforementioned, wrapped conditional classifier. On return of the REST call, the classification and confidence of the model are rendered in the UI. The user may confirm or correct the classifiers choice. The entered sentence and the optional user confirmation or correction is then stored in the backend in order to (1) display the five most recently entered sentences, (2) provide preliminary insight into the performance of the classifier on unseen sentences, and (3) preserve sentences for future training of the classifier. At this point, we support batch learning and plan to implement an online learning algorithm in future research to leverage the collected data directly for enhancing our conditional classifier. Our exemplary requirement is classified as Conditional Present with a confidence of 98.72 %. After confirming this correct classification, the user is forwarded to the second step.

Extraction of Conditionals In the second step, our pre-trained binary-file conditional extractor is loaded and used to annotate the entered sentence according to our fine-grained labeling scheme (see Figure 8.2b). The predicted labels per token are rendered in the UI. We provide an explanation of each label at the bottom of the UI to inform users about the meaning of the labels. For example, the expression *"the temperature change is requested"* is labeled as Antecedent 1. On the lower annotation layer, *"the temperature"* is labeled as Variable and *"is requested"* is labeled as Condition. Further, our extractor has correctly detected that Consequent 1

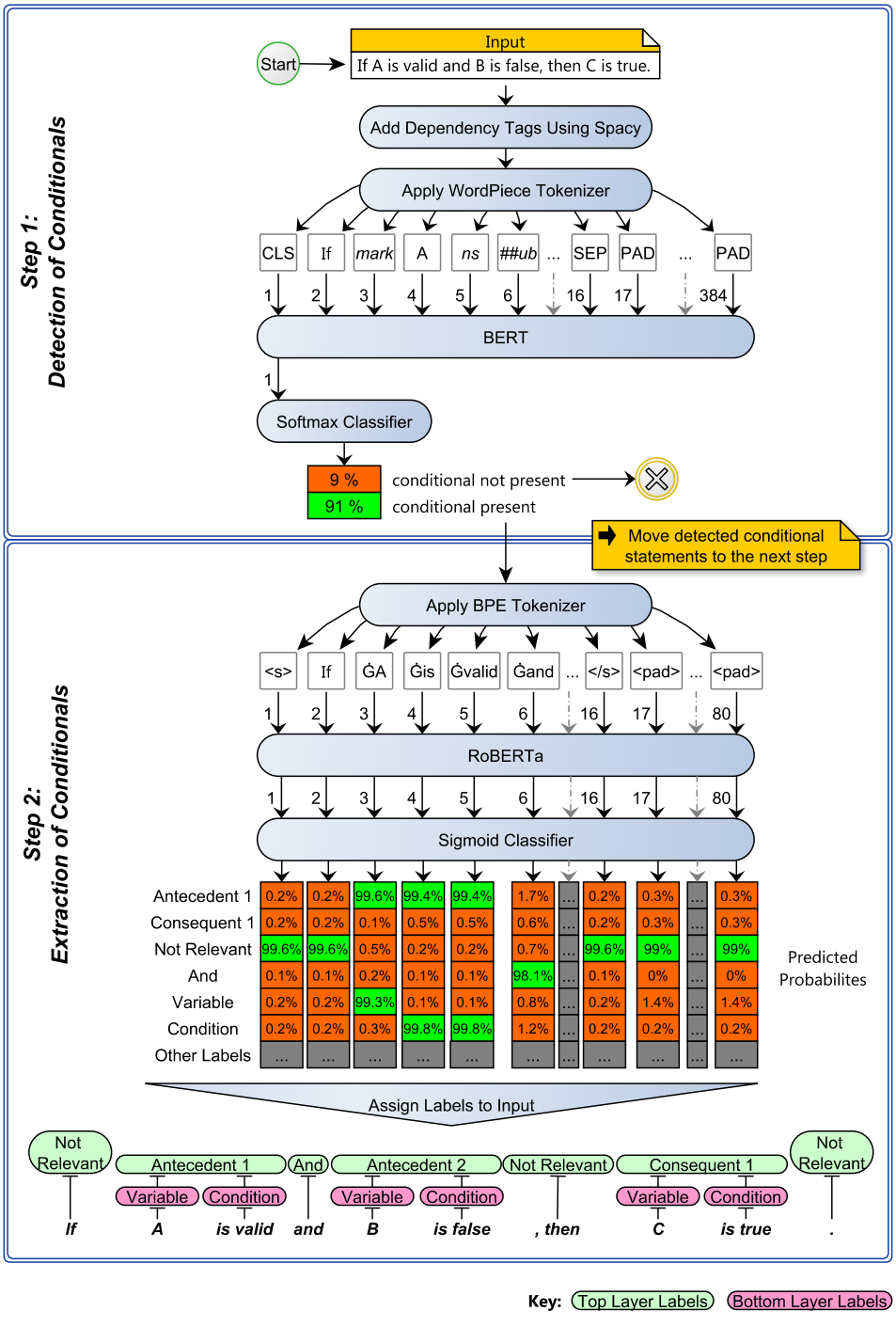


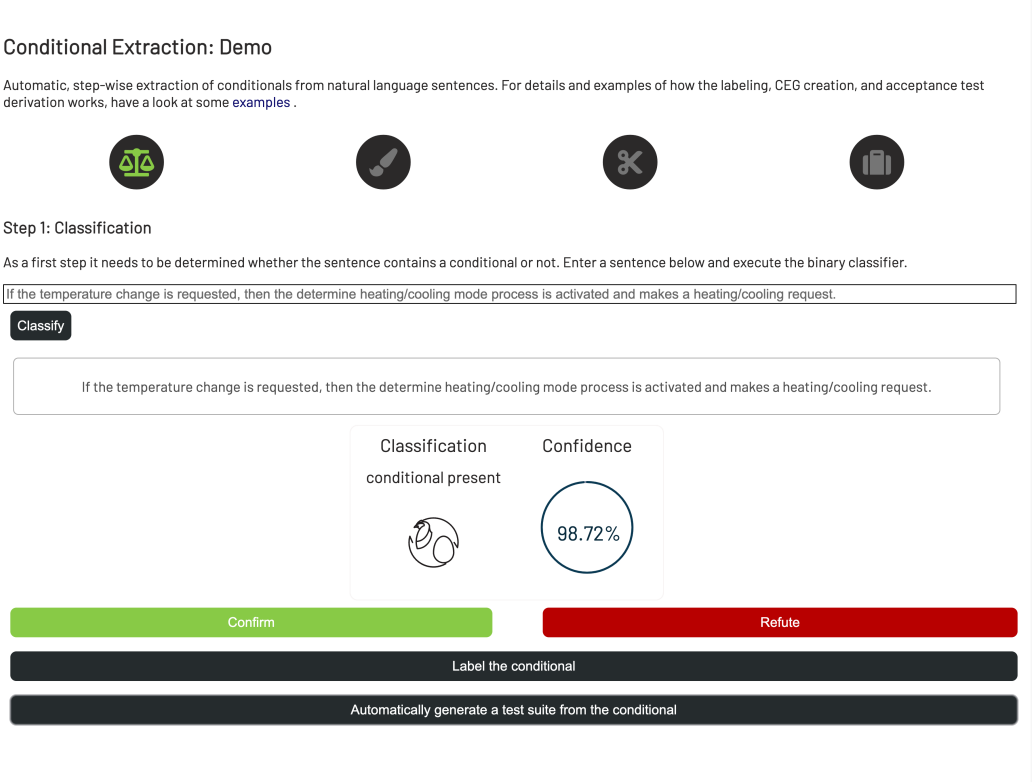
Figure 8.1: Overview of the CiRA Pipeline Consisting of Two Steps: (1) Detection of Conditionals, and (2) Fine-Grained Extraction of Conditionals.

and Consequent 2 are connected by a conjunction. In total, CiRA assigned nine labels to the entered sentence.

Creation of Cause-Effect-Graph In the third step, we create a CEG based on the annotated conditional (see Figure 8.2c). Specifically, we represent *antecedents* as cause nodes and *consequents* as effect nodes and relate them to each other using edges. The creation of the CEG is not a trivial, potentially error-prone task. To enable the user to correct potential errors manually or to modify the CEG for other reasons, we integrated a model editor into the tool. This allows users to add new nodes by means of simple drag and drop, or to adjust existing nodes and their edges. Elements can be removed from the CEG by pressing the DEL key. The auto-layout function supports the user in arranging the nodes to ensure clarity of the CEG.

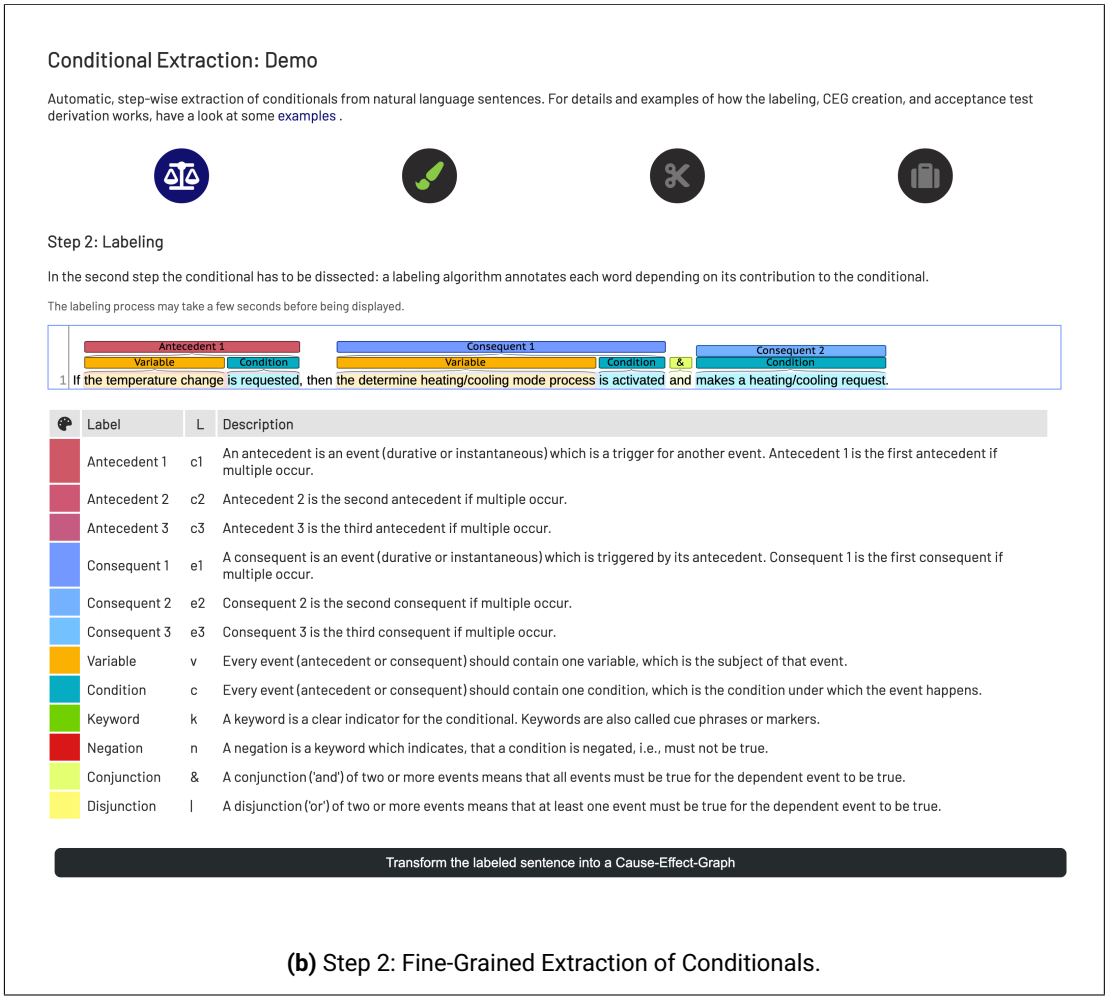
In the simplest case, *antecedents* and *consequents* encompass both a variable and condition in the lower annotation layer. We then fill the created cause and effect nodes with the corresponding information (see nodes representing Antecedent 1 and Consequent 1 in Figure 8.2c). If either of the two labels is missing, we need to extract the information from the nearest referent to correct incomplete nodes. In the given example, the variable of Consequent 2 is not included in the entered sentence. Hence, we enrich its corresponding effect node with the variable of Consequent 1 (see orange highlighting in Figure 8.2c).

Creation of Acceptance Test In the last step, we automatically derive the minimum number of test cases required to fully check the entered requirement from the created CEG (see Figure 8.2d). For this purpose, we consider the findings of our study on the logical interpretation of conditionals by RE practitioners. The user can choose whether s/he perceives *antecedents* to be both sufficient and necessary conditions for *consequents* or not (see checkbox below the test case specification). Depending on the selection, we filter the derived test cases and display the acceptance test that corresponds to the user's interpretation. In the given example, we perceive the *antecedent* as a necessary condition for both *consequents*. Accordingly, our approach derived two test cases from the created CEG.



(a) Step 1: Detection of Conditionals.

Figure 8.2: Overview of the User Interface Provided by `CiRA`.

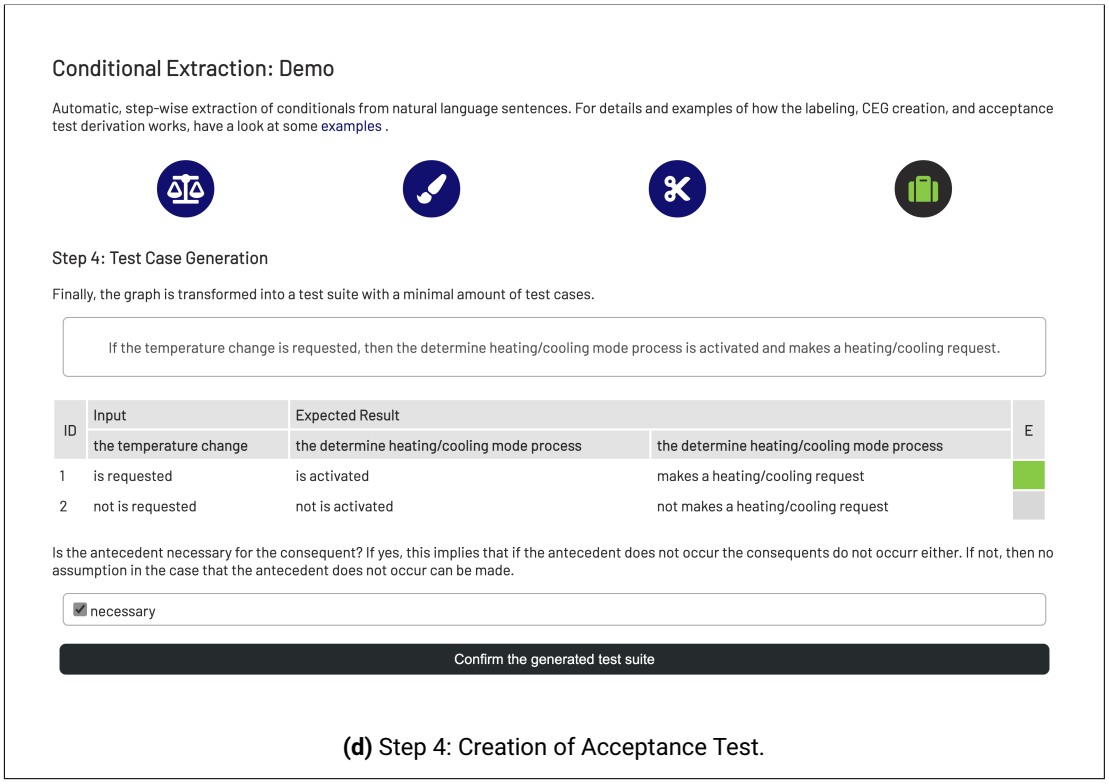


(b) Step 2: Fine-Grained Extraction of Conditionals.

Figure 8.2: Overview of the User Interface Provided by `CiRA` (cont.).

(c) Step 3: Creation of *Cause-Effect-Graph*.

Figure 8.2: Overview of the User Interface Provided by `CiRA` (cont.).



(d) Step 4: Creation of Acceptance Test.

Figure 8.2: Overview of the User Interface Provided by `CiRA` (cont.).

Part III

Leveraging Conditional Extraction for Automatic Acceptance Test Creation

Common Thread This part highlights how automatic conditional extraction can help to create acceptance tests. Specifically, we validate our claim from Section 1.2 that CiRA can facilitate the implementation of Use Case 1. We first motivate the use case by an industrial survey on challenges in agile testing (see Chapter 9). One of the most frequently cited challenges involved the creation of appropriate acceptance tests, which empirically substantiates the need for a method for automatic acceptance test creation. We address this challenge in Chapter 10 and present an approach capable of deriving acceptance tests from conditional statements automatically. Our approach leverages CiRA to extract conditionals in fine-grained form and then maps the extracted conditionals into a *Cause-Effect-Graph* that allows for the automated derivation of test cases. We evaluate the approach in a case study with three industry partners: *Allianz Deutschland AG* (insurance), *Ericsson* (telecommunication), and *Leopold Kostal GmbH & Co. KG* (automotive).

Preliminaries To understand this part, the functionality of CiRA needs to be thoroughly understood. Readers should therefore first read Part II, particularly Chapter 8. For the derivation of test cases from the extracted conditionals we build on an already existing model based testing technique: *Cause-Effect-Graphs*. To follow our approach presented in Chapter 10, readers should thus be familiar with the basics of *Cause-Effect-Graphing* (see Section 2.4).

Chapter 9

Agile Test Artifacts: Quality Factors and Challenges

Common Thread In Section 1.2, we claimed that we need a method that automatically derives the minimal set of required test cases automatically from NL requirements. In this chapter, we present an industrial survey with 18 practitioners from 12 companies operating in seven different domains that provides supporting evidence for this claim. In our survey, we analyze *how* agile test artifacts are designed in practice and *which* properties make them useful for quality assurance. Following the idea of *Activity-Based Artifact Quality Models* [308], we postulate that the quality of a test artifact depends on the stakeholder using it and the activities for which it is used. Accordingly, we explore properties (so-called *"quality factors"*) of test artifacts that have a positive or negative impact on the activities of the stakeholders. To understand why a certain property is considered good or bad by the practitioners, we first study current challenges in using test artifacts. Consequently, we distill a list of concrete factors describing what agile test artifacts should look like.

Contribution Our analysis reveals nine challenges and 16 factors describing the quality of six test artifacts from the perspective of agile testers. Interestingly, we observed many challenges regarding language and traceability, which are well-known to occur in non-agile projects. One of the most frequently stated problems concerned the lack of adequate acceptance tests. We found that acceptance tests are often not systematically created, resulting in incomplete or excessive test cases. In the case of missing test cases, system defects are not (or only partially) detected. In contrast, excessive test cases lead to unnecessary testing efforts and increased test maintenance costs. Consequently, practitioners need to strike a balance between full test coverage and number of required test cases. We also found that the creation of acceptance tests is a predominantly manual task due to insufficient tool support. These results emphasize the need for a tool-supported approach capable of deriving the minimal set of required test cases automatically from NL requirements.

Related Publications This chapter is taken, directly or with minor modifications, from a previous publication [3].

9.1 Research Objective

Background The *Agile Software Development* (ASD) principle *"working software over comprehensive documentation"* promotes that documentation should be kept to what is necessary or useful [309]. Hence, common ASD frameworks, such as *Scrum* [310], mention only few artifacts (*epics*, *user story*, etc.) that should be created, used, and maintained for documentation purposes. Instead, face-to-face communication should be encouraged in order to convey information. Nevertheless, agile practitioners have increasingly changed their attitude towards documentation [311] and are producing a variety of artifacts that are not inherent to ASD [312, 313, 314]. According to Wagenaar et al. [315], practitioners need additional artifacts for four reasons: i) they provide team governance, ii) they are useful for internal communication, iii) they are needed by external parties, and iv) they are useful for quality assurance. For the latter reason, a range of additional artifacts (e.g., *acceptance tests*) are commonly created to perform comprehensive software testing.

Research Goal Currently, we understand *which* test artifacts agile teams introduce (or should introduce) on their own initiative [312] and *why* they are needed [315]. However, empirical research on *how* agile test artifacts are designed in practice and, more specifically, *which* properties make them useful for quality assurance remains sparse. Existing normative standards such as the ISTQB *Acceptance Testing Syllabus* [316] or ISO 29119:2013 [317] occasionally mention some properties that test artifacts should possess. However, there are issues with these normative standards. Firstly, the list of properties is not complete—most of the properties are defined for the artifacts introduced by the ASD frameworks but not for the additionally-required artifacts introduced by the team. Secondly, normative standards describe quality through abstract properties—e.g., *acceptance criteria* should be both *"precise and concise"* [316]. The standard does not provide any further description of what is meant by these vague properties. Thirdly, the empirical basis and reasoning for these criteria remains unclear. This implies that the criteria are difficult, if not impossible to falsify.

We argue that for a combination of all of these reasons, we observe in practice that agile teams fail to satisfy these normative criteria, and struggle in maintaining their documentation artefacts [318]. In contrast to these existing ways to define quality criteria, we argue that quality of test artifacts should be defined from a quality-in-use perspective. Following the idea of *Activity-Based Artifact Quality Models* [308], we postulate that the quality of a test artifact depends on the stakeholder using it and the activities for which it is used. Accordingly, we **explore properties (so-called *"quality factors"*) of test artifacts that have a positive or negative impact on the activities of the stakeholders**. To understand why a certain quality factor is considered good or bad by the practitioners, we first study current challenges in using test artifacts. Consequently, we extend the normative qualities with a list of concrete factors describing what agile test artifacts should look like. In order to identify and understand the quality factors of agile test artifacts, we chose (qualitative) survey as our research method. For our study, we followed the guidelines by Ciolkowski et al. [285] for conducting empirical studies based on surveys. Following the *Goal-Question-Metric* [319] technique, we define the goal of our survey as follows:

- **Object:** Test artifacts

- **Purpose:** Identify, understand, and define
- **Focus:** Quality factors
- **Viewpoint:** Agile practitioners
- **Context:** Agile Software Development Projects

The expected outcome of our survey is a better understanding of quality factors of test artifacts. In our activity-based quality understanding, these are properties that positively or negatively affect the stakeholders and their follow-up activities. These quality factors should provide guidance for practitioners on how test artifacts should be designed. It should further establish the foundation for a systematic quality control of test artifacts. Based on the classification of Robson [270], our research goal is exploratory as we are seeking for new insights into the quality of agile test artifacts.

Research Questions Based on the idea that artifact quality is determined by the context in which it is used, we derived five research questions (RQ) from our survey goal. Each RQ addresses a specific component of the ABAQM meta model (see Figure 9.1).

- **RQ 1:** Which stakeholders are involved in agile testing?
- **RQ 2:** Which activities are performed by the stakeholders?
- **RQ 3:** Which artifacts are used by the stakeholders in the context of these activities?
- **RQ 4:** Which quality factors positively influence the execution of these activities?
- **RQ 5:** Which quality factors negatively influence the execution of these activities?

9.2 Survey Design

Theoretical Foundation *Activity-based Artifact Quality Models* (ABAQM) are based on the idea that it is not sufficient to speak of *good and bad quality* in general since the quality of an artifact depends on the context in which it is used [308]. More specifically, quality is determined by the stakeholders and the activities that they conduct with the artifact. The quality of an artifact is considered good if its properties allow stakeholders to effectively and efficiently carry out their activities. In this chapter, we create an ABAQM for all test artifacts involved in ASD, which enables us to understand their quality in the agile context. We use the following concepts to describe an ABAQM (see a meta model in Figure 9.1):

Artifact Following the quality-in-use paradigm, an artifact is a collection of coherent documented information which assists a stakeholder in reaching the project goals. Examples of artifacts are *use case documents* and *test data*. Artifacts that share similar properties can be combined into a generalized super-class. For example, *unit tests*, *integration tests*, and *system tests* address different test levels but can be bundled into a super-class Test. In addition, artifacts can contain other artifacts, such as a *user story* which contains multiple *acceptance criteria*.

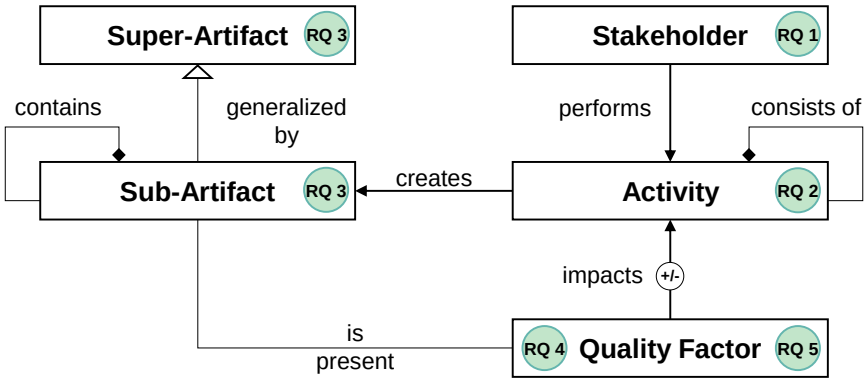


Figure 9.1: ABAQM Meta Model and Mapped RQs, Based On [35].

Stakeholder A stakeholder is interested in an artifact and uses it during a certain activity. An example of a stakeholder is a *test designer*, who uses *user stories* to derive *acceptance tests*.

Activity An activity is an invested effort which involves one or more of the mentioned artifacts. An activity can be divided into sub-activities. For example, *acceptance test design* can be decomposed into *acceptance test creation* and *acceptance test updating*. During an activity, stakeholders do not only use artifacts but also create new ones. Hence, artifacts can be both input and output of activities.

Quality Factor A quality factor is a property that is or is not present in an artifact. Femmer and Vogelsang stress that this property *"must be objectively assessable through a measure to be used for quality control"* [308]. For example, a *test* should only contain the minimum number of required *test cases* to avoid excessive testing. This quality factor *minimal* can be evaluated objectively.

Impact An impact is a relation between a quality factor and an activity. The relation can be either positive (i.e., the presence of the quality factor supports the stakeholder in the execution of an activity) or negative (i.e., the quality factor hinders the stakeholder). The aforementioned quality factor *minimal*, for example, has a positive impact on the activity *testing*.

Population and Survey Sample The selection of the survey participants was driven by a purposeful sampling strategy [320]. Specifically, we defined criteria that the participants need to meet to be suitable for our survey, a) they work for a company that develops software following a defined agile software paradigm (e.g., *Scrum*, SAFe), b) they have been involved in the testing process for at least one year, and c) they create, use and/or maintain at least one test artifact. Each researcher involved in the survey prepared a list of potential interview partners using their industrial contacts (convenience sampling). From this list, the research team jointly selected suitable partners based on their adequacy for the study. To further increase the sample size, we asked each interviewee for relevant contacts after the interview (snowball sampling). We stopped conducting more interviews after we reached saturation. More precisely, once we could no longer identify new quality

Table 9.1: List of Participants.

Company	No.	Role	Size	Domain
C1	P1	Product Owner	150k	Insurance
	P2	Test Designer		
C2	P3	Test Designer	1k	Retail
	P4	Test Lead		
C3	P5	Test Lead	20	Software
C4	P6	Agile Team Lead	50	E-Mobility
C5	P7	Partner	500	IT Consulting
	P8	Software Developer		
	P15	Software Developer		
C6	P9	Product Owner	10	PropTech
C7	P10	Agile Coach	50	IT Consulting
C8	P11	Agile Team Lead	1k	Software
C9	P12	Software Developer	2k	Software
C10	P13	Software Architect	200	Software
C11	P14	Software Architect	200	Software
	P16	Software Architect		
C12	P17	Business Analyst	40k	Reinsurance
	P18	Business Analyst		

factors. Table 9.1 presents an overview of the participants, their roles, and information about their companies. In total, 18 practitioners from 12 different companies operating in seven different domains participated in our survey. We did not restrict our population with regard to company size or application domain. Rather, we involved practitioners from companies of different domains and sizes to obtain a holistic understanding of test artifact quality.

Data Collection We chose interviews over other data collection instruments for two reasons. First, ambiguities in the questions can be resolved directly ensuring that all questions are understood correctly and that they are not skipped. Second, the interviewer can observe the behavior of the participants and ask them to elaborate their responses (e.g., to better understand the reasoning of the participant or to go deeper into details). This is particularly important to understand why the participant considers the quality of a certain test artifact good or bad.

Questionnaire Design Prior to conducting the interviews, we developed an interview guideline to gather the data for answering our RQs. We designed the interview questions to systematically identify the elements of the ABAQM to shed light on the quality of test artifacts. For this purpose, we followed the guidelines of Dillman et al. [289] to reduce common mistakes when setting up a questionnaire (e.g., avoiding double-barreled

Table 9.2: Questionnaire Structure.

intro	How many employees work at your company?
	In which domain does the company operate?
	Since when does your project follow the agile software paradigm?
	Which framework (*Crystal*, *Scrum* etc.) do you follow?
core	What is your role in the testing process?
	Which activities do you perform?
	What is the purpose of your activities?
	Which artifacts do you create as part of your work?
	Which artifacts do you use as part of your work?
	Which artifacts do you maintain as part of your work?
	What do you need the artifacts for?
	Do problems or challenges arise during your activities?
probing	What exactly about the artifact bothers you?
	How should the artifact be designed instead?
	How is the quality of the test artifacts currently checked?

questions). Since our research goal and RQ are of exploratory nature, most questions are open-ended. Our questionnaire consists of 15 questions, including 13 open-ended questions and two closed questions (see Table 9.2). In each interview, we asked introductory questions to gather information about the participant's background (e.g., company, experience in ASD), followed by questions about the activities of the participants and the artifacts they use in the context of these activities. To avoid misinterpretations, we gave a short briefing of the concepts central to this study at the beginning of each interview.

The greatest challenge in compiling the questionnaire was to develop questions for determining the quality factors. For this purpose, we discussed two different questioning strategies. First, ask the participant directly which quality factor are important for the artifact to be useful (e.g., *"which properties should the artifact possess from your perspective?"*). In this case, the participant is explicitly asked to state the quality factors. Second, initially ask the participants which problems occur during their activities and the usage of their artifacts. Subsequently, use probing questions to ask what exactly bothers the stakeholder about the artifact and how the artifact should have been designed instead. Using this *"problem-oriented questioning approach"*, we first determined the problems related to artefact usage and then derived the respective quality factors from these problems.

Pilot To decide which questioning strategy is best suited for the creation of a ABAQM, we designed a questionnaire for both strategies and evaluated them in a pilot (see step ① in Figure 9.2). We conducted the pilot phase iteratively; it consisted of two parts, the internal pilot and the real case pilot. In the internal pilot, the questions were continuously refined by the research team with regard to suitability, understandability, and correctness. The real case pilot involved two interviews with participants from

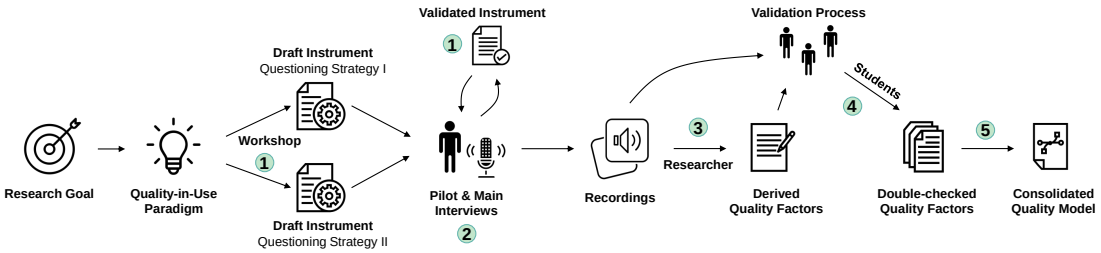


Figure 9.2: Overview of the Method Followed in Our Industrial Survey: (1) Preparing and Validating the Instrument, (2) Conducting Interviews, (3) Qualitative Content Analysis, (4) Review Process, and (5) Creating the *Activity-Based Artifact Quality Model*.

the targeted population, and revealed that the *"problem oriented"* approach is more suitable for collecting the quality factors. Practitioners struggle to abstract and define independently which property of an artefact leads to good or bad quality. It proved more effective to gather the quality factors together by first discussing current challenges and then successively deriving the quality factors of the artifacts. Hence, the probing questions are an integral part of our questionnaire as they encourage the participants to expand a particular anecdote and to define precisely what they like or dislike about the artifact.

Survey Implementation and Execution During this step, we compiled all the material needed to conduct the survey. We prepared an invitation letter to ask potential participants for an interview. In addition, we provided our questionnaire to the participants in advance to allow them to get a first impression of the content of the interview and prepare accordingly. All interviews were conducted by the first author. The duration of the interviews had an average of 41 minutes with a minimum of 31 and a maximum of 67 minutes. They took place from March to May 2020. All interviews were conducted remotely via *GoToMeeting* and *Google Hangouts* as face-to-face interviews were infeasible due to COVID-19. We interviewed all participants individually to prevent that their statements were influenced by others. The participants were informed, before starting the interview, that the data will be treated anonymously. Additionally, all interviews were conducted in the native language of the participant—German, with the exception of one in English. The audio of all interviews was recorded with the permission of the participants for subsequent analysis (see step ② in Figure 9.2). Due to confidentiality agreements with the respective individuals, the recordings cannot be published.

Survey Analysis and Packaging Since most of our interview questions are open-ended, we decided to use qualitative content analysis in order to analyze the interview data (see step ③ in Figure 9.2). Following the guidelines of Mayring [321], we conducted a content analysis inductively as our research goal is explorative and we needed to derive the quality factors of the test artifacts from the interview data. The first author analyzed the interview recordings and performed two steps for each interview. First, for each artefact discussed in the interview, the mentioned problems were determined. Second, we derived factors from theses problems that influence the quality of the artifacts positively or negatively. In particular, we studied the answers to the probing questions, which give a precise insight into which quality factors are considered good or bad. To

validate our results, we performed an internal review process (see step ④ in Figure 9.2). We involved three students and provided them with the interview recordings as well as the hypotheses and derived quality factors. They performed the same steps as the first author and compared the results in order to agree on the information to be extracted. In case of deviations, the respective passages in the interview were analyzed together until reaching a consensus. After the validation process, we used frequency analysis to find out which problem was mentioned more often. This provides a first indication of where systematic quality control may be most needed. We report our results in two ways. Firstly, in the form of a research paper to share our findings regarding quality control of agile test artifacts in the research community. Secondly, as an executive summary to share the results with the interviewed practitioners.

9.3 Survey Results

This section presents the results of our survey structured according to the research questions. Based on our activity-based quality understanding, we describe for each test artifact (see **RQ 3**) the stakeholders using it (see **RQ 1**) and the context of the activities (see **RQ 2**). As described in the previous section, we applied a *"problem oriented"* questioning approach to determine the quality factors. We report the identified challenges arising when the stakeholders use each artefact during their activities. From these challenges, we derived the factors that positively (see **RQ 4**) or negatively (see **RQ 5**) influence the quality of the artifacts. A positive impact of a quality factor is indicated by ⊕, while a negative impact is indicated by ⊖. The artifacts that were discussed by our participants were *acceptance criterion*, *acceptance test*, *feature*, *test documentation*, *test data*, *unit test* and, finally, *all test artifacts* for factors independent of the concrete artifact. Our final ABAQM (see Figure 9.3) includes 16 quality factors. Most of the quality factors support the stakeholders in carrying out their activities (13 out of 16 quality factors). However, three quality factors hinder the stakeholders in performing certain activities. We argue that future agile test artifacts need to be designed according to these quality factors in order to make them useful for the stakeholders.

Artifact 1: Acceptance Criterion

Acceptance Criteria (ACC) are conditions that a system must meet in order to fulfill a user story and be ultimately accepted by the user. ACC are used by test designers during acceptance test creation. This activity involves two steps. Firstly, the test designer analyzes all ACC assigned to a particular user story to understand the expected system behaviour. Secondly, the test designer derives test cases for each ACC and merges them into an acceptance test, which is later used in the quality assurance process to check the compliance of the system with the user story. All test designers interviewed stated that both steps are performed manually. We found two major challenges with the usage of ACC during *acceptance test creation*. From these challenges, we derived six quality factors.

Challenge 1: Acceptance Criteria Are Ambiguously Formulated. The interviewed participants complained about the poor linguistic quality of the ACC. In many cases, it is not clear *"what exactly the system is supposed to do, which makes it difficult to derive*

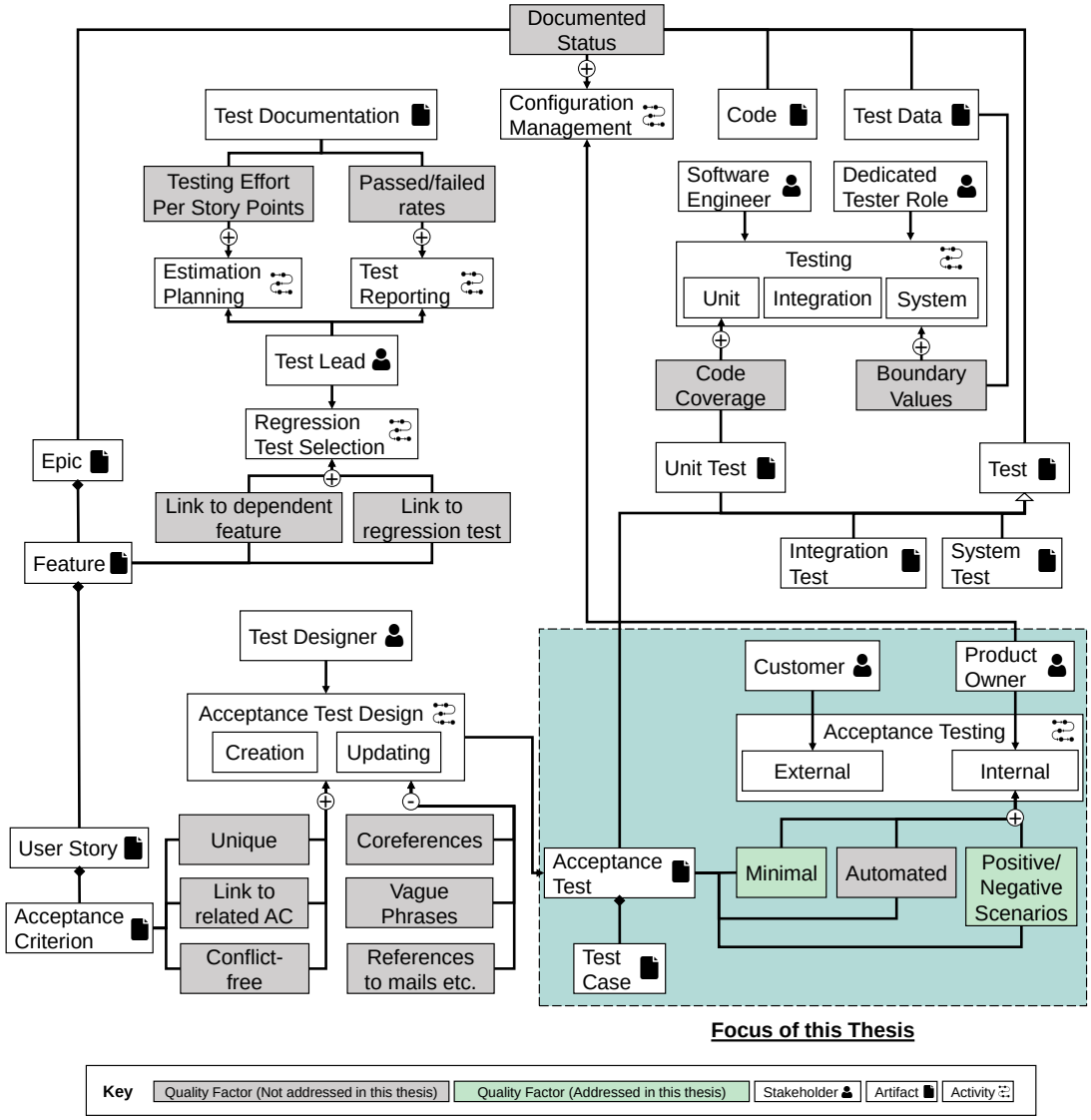


Figure 9.3: Activity-Based Quality Model for Agile Test Artifacts. In This Thesis, We Propose an Approach Capable of Automatically Creating Acceptance Tests in Compliance With the Two Quality Factors **Positive and Negative Scenarios** and **Minimal**. The Focus of Our Thesis Is Highlighted by a The Focus of Our Thesis Is Highlighted by a Green Frame.

test cases" (P3). Hence, *test designers* need to contact the *business analyst* who defined the criterion and clarify its meaning before they can start with the actual test case creation. This leads to delays in the test design process. According to P2 and P3, it would be helpful if the formulation of an ACC is checked prior to the test process to ensure that only testable ACC are submitted to the *test designer*. For this purpose, the *test designer* should be involved in the formulation of the ACC. However, this would require additional resources which are very limited in practice as stressed by P2 and P17:

> *"We lack the time to discuss every acceptance criterion with each other. We are* ***dependent on the formulation skills*** *of our business analysts."* (P2)

> *"The quality of acceptance criteria varies from project to project. Some analysts specify them precisely, while some do not. However, checking the formulations* ***manually is not possible*** *due to tight time constraints."* (P17)

Hence, a quality assurance check of *acceptance criteria* should be performed automatically to be suitable for practical use. In this context, the ACC should be reviewed with respect to the following quality factors:

QF 1: Coreferences ⊖ A *user story* usually contains multiple ACC specifying the expected system behavior. In practice, these ACC often contain coreferences (i.e., expressions that refer to the same entities). As the number of ACC increases, it becomes difficult to resolve these coreferences correctly, which hinders the test design. Hence, ACC should not contain coreferences to ensure testability.

QF 2: Vague Phrases ⊖ The interviews show that ACC are usually defined using *Unrestricted Natural Language*. The use of NL is intuitive for *business analysts*, but bears the risk of vagueness and ambiguity. As already stated by Berry and Kamsties, this can lead to *"diverging expectations and inadequate or undesirably diverging implementations"* [322]. We found that vague phrases often occur in ACC and hinder the *acceptance test design*.

> "*You often see criteria like "the system should be able to upload the data quickly." What exactly is meant by quickly? You do not know then* ***what to test***." (P11)

> "*A typical example you often see: "if possible, the system should do xy." It is unclear* ***what possible means***." (P2)

We refrain from listing all vague phrases in the ABAQM since a number of studies (e.g. [323, 35]) have already dealt with this quality factor in requirements. Instead, we want to explicitly point out that this quality issue is also relevant for ACC.

QF 3: References to Emails, Calls, and Documents ⊖ Instead of fully documenting the desired system functionality, expressions like *"as discussed by phone"* are often found within the ACC. This leads to a series of problems. First, the ACC can only be understood by the stakeholders involved in the call and cannot be converted into test cases by another *test designer*. Therefore, the testability of the ACC is limited due to the undocumented, implicit knowledge. Second, information about the system functionality is lost with changes in the project team and it is no longer known which functionality the created test case was initially supposed to test. As a result, there is no traceability between the created *test case* and the ACC. In C1, ACC also often

contain references to other *acceptance criteria* or documents (e.g., *"as described in document x"*). Due to the high change dynamics in agile projects, these references become quickly outdated, which results in gaps in the requirement specification.

Challenge 2: Lack of an Overview of Dependencies Between Acceptance Criteria *Acceptance test design* involves not only the creation of *acceptance tests* for new system functionalities but also the adaptation of existing *acceptance tests* to changing customer requirements. The latter is essential in order to keep requirements and tests aligned. For this purpose, *test designers* need to understand the relationship between already implemented *user stories* and new *user stories* and adapt the test suite accordingly. If the new *user story* introduces a new functionality, a new *acceptance test* must be created and added to the test suite. If the *user story* changes an already implemented functionality, the existing *acceptance tests* must be adapted. However, it is increasingly difficult to identify these relationships due to the high number of *user stories*. The interviews showed that for every requested change in the software a new *user story* is created, rather than the existing *user story* changed. This observation coincides with the results of the study by Hotomski et al. [318]. Consequently, the number of *user stories* and ACC is growing steadily, making it more and more difficult to keep track of them:

> "*If someone adds a new user story to the backlog that changes or overwrites another user story, we don't notice it. So we* ***don't know which tests we need to adjust***." (P2)

Instead, separate *acceptance tests* are created for each *user story*, resulting in a test suite constantly increasing in scope and complexity. When the test suite is executed and some tests fail, "*we don't know if these tests reveal a real bug in the system or if they are checking old functionality and should have been updated*" (P2). This leads to additional effort and therefore high testing costs. A similar situation is found in company C2:

> "*We do not know whether some acceptance criteria* ***overlap or even contradict each other***. *Hence, it sometimes happens that we create contradictory test cases.*" (P3).

QF 4: Conflict-Free ⊕ A *test designer* can only maintain a consistent test suite if the underlying ACC are not contradictory. Consequently, practitioners require a method that automatically compares ACC with each other and reveals inconsistencies. This will have a positive impact on both *acceptance test creation* and *updating* as it indicates which *user stories* the *test designer* needs to check, as stressed by P2:

> "*As a test designer I have to understand where and how the functionality is supposed to change and how I need to adapt my tests. If you could somehow* ***automatically display overlaps between acceptance criteria****, that would help me a lot.*" (P2).

In the case of major changes introduced by the new *user story*, the existing *acceptance test* should be archived and a new *acceptance test* created. Otherwise, the existing *acceptance test* should be adapted. This helps to avoid *False Negatives* (i.e., invalid failing tests) during test execution. According to P2, P10, and P11, the old *user story* should be eventually assigned the status *"old"* and linked to the new *user story*. This is essential to enable version control of the test assets as described in Section 9.3.

QF 5: Unique ⊕ In order to prevent the creation of unnecessary tests, it is essential that ACC do not describe redundant functionalities. If two ACC describe the same functionality, the *business analyst* needs to be informed and both ACC need to be merged.

QF 6: Link to Related Acceptance Criteria ⊕ *Acceptance tests* sometimes cover more than one ACC, sometimes of more than one *user story*. Therefore, knowing the relations between ACC enables to minimize the overall testing effort since related functionalities can be tested simultaneously.

> "*We often noticed afterwards that test cases* ***could have been bundled together****, e.g. for acceptance criteria that describe the same UI view.*" (P17)

> "*We've been trying to optimize our test suite for some time now. But we fail frequently to create combined test cases for related requirements, because we* ***do not know which acceptance criteria belong together****.*" (P4)

In order to create such joint tests, the *test designer* needs to understand the relationships between the ACC and consider them during test case creation. Accordingly, the quality of ACC is considered good if they are linked to related ACC. An example are two ACC that handle the same input parameters as ACC 1: *"If input A then function B"* and ACC 2: *"If input A and input B then function C."* Both ACC can be checked with a joint *acceptance test*.

Artifact 2: Acceptance Test

Acceptance tests are instruments used to verify the conformity between user expectations and actual system behavior (*acceptance testing*). Each *acceptance test* contains a set of *test cases*. In practice, there are two ways of *acceptance testing*. Internal *acceptance testing* (alpha testing), which is performed by members of the organization that developed the software, and external *acceptance testing* (beta testing), which is performed by the customer. As we were not able to talk to customers during the study, we examine the quality of *acceptance tests* from the perspective of an internal *product owner* in the context of alpha testing. *Product owners* utilize *acceptance tests* to verify whether the developed system complies with the requirements and can be delivered to the customer. Our interviews reveal two challenges with the usage of *acceptance tests* during *acceptance testing*. We derived three quality factors from the identified challenges.

Challenge 3: Acceptance Tests Contain Too Many or Too Few Test Cases We found that *acceptance tests* are often not systematically created resulting in incomplete or excessive *test cases*.

> "*We do not follow any particular procedure in the preparation of acceptance tests. Every test designer does this based on his experience. Of course, such a manual process is prone to errors, because you can* ***overlook some cases****. In fact, I have also been in the situation where I forgot test cases.*" (P3)

In the case of missing *test cases*, system defects are not (or only partially) detected. As a result, faulty software is ultimately delivered to the customer, leading to errors in production and lower customer satisfaction. According to an internal analysis conducted

by company C1, 83 % of the system defects at company C1 could have been detected by more complete *test cases*. A similar observation was made in company C4:

> "*We often experience that our live system does not completely fulfill all user stories. This could have been avoided by the* ***right acceptance tests***" (P6).

Instead of systematically determining which *test cases* are required to cover an ACC, they are usually created based on past experience of the *test designer*. This makes the test case derivation error-prone and increases the risk of missing test cases, as *test designers* "*tend to only test the positive cases and not the negative ones*" (P1, P9). A major challenge is the complexity of the ACC:

> "*We often have to implement highly complex business rules that include a range of parameters. It's hard to decide* ***which combinations of parameters should be tested***" (P1).

This increases the risk of missing *test cases*. However, not only *test cases* are missing, but also superfluous *test cases* might be created leading to an increase in the testing effort. According to P1, many *test designers* are lacking the required qualification and, more importantly, the time for a systematic test case derivation. Consequently, there is a great demand for an automated test case derivation from ACC to maintain the high development speed.

QF 7: Positive and Negative Scenarios ⊕ *Acceptance tests* are only suitable for detecting system defects if they are complete (i.e., covering all positive and negative *test cases*).

QF 8: Minimal ⊕ Achieving QF 7 is crucial to the quality of an *acceptance test*, however, it is also necessary to strike a balance between full test coverage and the number required of *test cases*. More specifically, an *acceptance test* should contain only the minimum number of *test cases* needed to fully cover the ACC in order to minimize the required testing effort.

Challenge 4: Lack of Automation of Acceptance Tests The interviews revealed that the degree of test automation is still insufficient in practice. At the lower levels of the test pyramid, such as *unit tests* and *integration tests*, the execution has mostly been automated. However, *acceptance tests* are usually performed manually resulting in large testing efforts.

> *"Our acceptance tests are* ***always carried out manually****. Therefore, the testing process takes a rather long time and we are highly dependent on how the product owner performs the test."* (P18)

This represents a major challenge, especially as the development project proceeds and the system's functionality increases:

> *"Before each new release, we have to run acceptance tests that check already implemented user stories to avoid regression. The number of tests quickly increases during a project and then you ask yourself* ***who is going to execute these tests****? We have to run the new acceptance tests as well."* (P7)

The reason for the low automation of *acceptance tests* stems not from limited tool support, but rather from the fact that many companies still neglect to use them: We found that some smaller companies like C6 have already automated the majority of their *acceptance tests*. The problem of insufficient automation occurs mainly in large companies like C1 and C2. According to P9, P1 and P6, this might be due to the culture of these companies, who allegedly refuse to implement new automation tools initially and therefore introduce them with a considerable delay.

QF 9: Automated ⊕ In order to cope with the high development speed, *acceptance testing* needs to be automated, e.g. through tools such as *Selenium*, *Cypris*, and *Robot Framework*.

Artifact 3: Feature

A *feature* is a specific piece of functionality that is desired by the customer. In case of new or changed *features* in the current iteration, the *test lead* must verify that no regression on already implemented *features* is introduced (*regression testing*). With a growing system, the scope of *regression testing* also increases, so that running an entire regression test suite is time consuming:

> *"In our project, some manual regression tests* ***take four days****."* (P4)

A similar picture emerges with automated regression tests performed by continuous integration tools:

> *"We run our automated tests via Travis. The* ***Travis build*** *for our entire application* ***takes an entire day****."* (P5)

Such long test suite runs pose a major problem, especially considering the short sprint cycles that are often only two weeks. Selecting the right regression tests is therefore essential in order to minimize the testing effort. Specifically, the *test lead* needs to run the regression test for the changed *feature* and for all dependent *features* to identify potential regressions (*regression test selection*). For this purpose, knowledge about *feature* dependencies is required. We found one major challenge related to the usage of *features* during *regression test selection*. From this challenge, we derived two quality factors.

Challenge 5: Lack of an Overview of Dependencies Between Features In practice, there is no overview of the relationships between *features*. As a result, there is a negative impact on the *regression test selection* as it is not transparent which regression test runs are necessary. Consequently, practitioners need to test on a risk-based basis:

> *"I execute the regression tests of all those features which I know from* ***experience*** *to be related to the changed feature."* (P5)

This is prone to errors resulting in a growing desire of the *test leads* for a *"functional structure"* in their project, which indicates the relationship between the *features*. This allows to understand which features might be impacted by a change of a certain feature and need to be tested.

QF 10: Link to Dependent Feature ⊕ According to P4 and P5, there are too many *features* in the projects to manually track the dependencies between them. Furthermore, existing tools such as *Jira* do not provide the option of illustrating relationships between *features* via links. Hence, there is a need for a method that automatically reveals dependent *features* in order to establish the *"functional structure"*.

QF 11: Link to Regression Test ⊕ For the selection of regression tests, the *test lead* requires a clear traceability between *features* and corresponding regression tests. Hence, each *feature* must have a link to a corresponding regression test.

Artifact 4: Test Documentation

In addition to *regression test selection*, the *test lead* is also responsible for *test reporting* and *estimation planning*. As part of *test reporting*, the *test lead* has to provide an overview of successful and failed tests after each iteration to track the progress of the development team. For this purpose, a comprehensive *test documentation* is required. The *test documentation* contains information about the testing team's progress, achieved results, and overall testing strategy. We found one major challenge related to the usage of *test documentation* during both *test reporting* and *estimation planning*. From this challenge we derived two quality factors.

Challenge 6: Test Results and Effort Are Not Properly Documented The interviews revealed that there is a common problem that test results are not properly documented at all test levels. Especially on the intermediate test levels such as integration testing there is a lack of an overview of the results. Therefore the reporting is mainly done on unit and acceptance level.

> *"I experienced a number of projects that do not separate between the test levels and consider every technical test as a unit test and simply document* ***all results at unit level****."* (P10)

Thus, it is difficult to identify the bugs at the correct test levels and change the software accordingly.

QF 12: Contains Passed/Failed Rates at Each Test Level ⊕ Each test type needs to be linked to its respective test result. Specifically, the *test documentation* needs to contain the corresponding test result for each *unit test*, *integration test*, *system test*, and *acceptance test* to provide a comprehensive overview of all testing levels at any time during the life cycle of the software.

In addition, we found that not only the test results, but also the test effort is not properly documented, which leads to issues in *estimation planning*. This activity aims at planning the resources needed to perform the testing in the next iteration:

> *"It is very difficult to estimate the required number of testers when I join a new project as there is* ***no documentation of the required test efforts*** *from previous iterations."* (P4)

Thus, there is no indication how much working days were needed by the testers involved in the project to validate former *user stories*, as stressed by P4 and P13:

*"We need some **key performance indicators** especially for agile testing that are tracked and documented during the test execution. Based on these, I can plan future test activities."* (P4)

*"For example, it would be great to know **how many story points** were implemented and tested in past sprints."* (P13)

QF 13: Contains Testing Effort per Story Point ⊕ A *test lead* needs an overview of the number of *user stories* implemented in past sprints, their story points and the required test effort. Specifically, it is necessary to document how many working days the testers needed per story point to get an overview of their testing capabilities and estimate future testing efforts accordingly.

Artifact 5: Test Data

In addition to suitable *test cases*, *test data* is also needed to systematically test the behavior of the system. *test data* is therefore required at all test levels. Our interviews indicated that there are two main approaches to decide which stakeholder is responsible for which testing level. In small companies, the *software engineer* is usually conducting end-to-end tests, i.e. he is responsible for all test levels:

*"We do not have dedicated tester roles. Our software engineers perform the **entire process** from unit testing to system testing."* (P9, P6)

In large companies, the test levels are allocated to different roles:

*"Unit tests are written and executed by our software engineers, but we have **different testers** who perform integration tests or other test types."* (P18)

However, the interviews showed that all roles have an interest in high quality *test data*. We found one challenge related to the usage of *test data* and derived one quality factor.

Challenge 7: Lack of Test Data to Properly Test the Software In practice, the generation of *test data* is a great challenge, so that *"we often do not have enough test data or the quality of our test data is poor"* (P1). In this context, poor quality denotes the deviations from real production data. The participants complained that their *test data* often does not cover all possible boundary cases that might occur in production, so that the system is not tested under all potential conditions. We found this issue in large companies like C1 as well as in small companies like C6. This is mainly caused by the fact that *test data* is not systematically derived from production data, but rather that testers use random test values as *test data*.

*"I often see the problem in projects that poor test data is used. Poor means that the developer enters **arbitrary values** in his unit test, but omits constellations that might occur in practice. Obviously, this leads to errors."* (P7)

Hence, the testing of all possible boundary values depends on the experience of the testers.

QF 14: Boundary Values ⊕ To ensure that all potential exceptional conditions are covered during testing, the *test data* must contain the same boundary cases as the production data.

> *"We need a method that learns to generate **appropriate test data** from production data."* (P1)

In this context, it is crucial to keep the *test data* anonymized, especially when dealing with sensitive data.

Artifact 6: Unit Tests

Unit tests are usually implemented and used by *software engineers* to test individual units of the source code or sets of modules. We found one challenge related to the usage of *unit tests* and derived one quality factor.

Challenge 8: Inadequate Code Coverage of Unit Tests The interviews demonstrated that there are not only problems with functional testing (i.e., poor *acceptance testing*) but also problems with testing at lower test levels:

> *"There are always bugs in production that could have been detected **at lower levels**."* (P1)

The core problem mentioned by the participants is that *unit tests* are not created following a certain pattern. Rather, it depends on the developer and the reviewer which *unit tests* are created. This leads to a strongly fluctuating quality of the *unit tests*. Similar to the automation of *acceptance tests*, we found differences between small and large companies. The smaller companies were able to give us an overview of the code coverage of their *unit tests*, while larger companies were not aware of the quality of their *unit tests*.

QF 15: Code Coverage ⊕ To control the quality of *unit tests*, code coverage metrics should be applied as they allow to determine how much of the developed code is tested. The participants mentioned arbitrary thresholds (e.g., 80 %) which they considered useful.

> *"This does not mean that no errors can occur. But it provides a good **first overview of the quality** of my unit tests."* (P7)

All Test Artifacts

In the following, we present a challenge that applies to all test artifacts equally. Hence, the quality factor derived from this challenge is relevant for all presented test artifacts.

Challenge 9: Missing Version Control of All Test Assets *Configuration management* is an integral activity to monitor and control the status of software during its life cycle. Version control of source code and automated tests is already anchored in today's business practice and is supported by control systems such as *Git*, allowing to track code changes over time. However, version control of all agile test artifacts is only partially implemented:

> *"We often don't know what the software is capable of doing or has done at a certain point in time and **what exactly we have tested**."* (P1)

This poses a problem especially in regulatory environments. For example, companies in the insurance industry need to document the functionality of the different software versions and prove which tests have been performed to verify that functionality. Hence, practitioners *"need the historic information of all test assets."* (P1, P2).

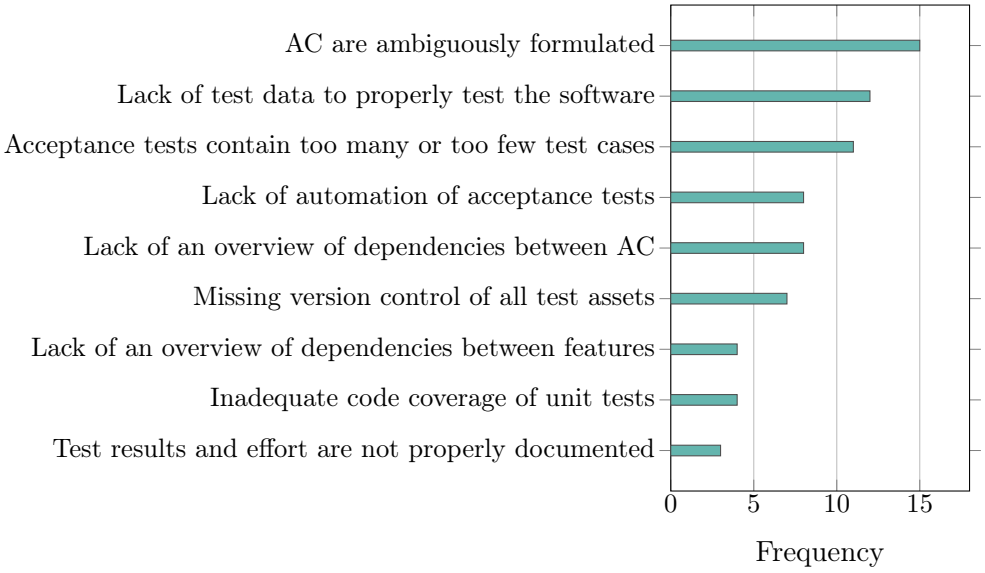


Figure 9.4: Frequency Analysis of Mentioned Challenges.

QF 16: Documented Status ⊕ All test artifacts need to be maintained with an appropriate status. This can be illustrated by a *user story* and its corresponding *acceptance test*. After the creation of a *user story*, its status is set to *"New"*. It will be set to *"Committed"* by the developer once its implementation has started. After the implementation, the *user story*'s status is set to *"Resolved"* and is finally set to *"Done"* by the *product owner* if the *acceptance test* was successful. To monitor the status of the software during its life cycle, it is indispensable that artifacts are archived and not discarded. For example, if a new *user story* overwrites another one, the old *user story* including the *acceptance tests'* status should be set to *"Old"* and the old *user story* should reference the new *user story*. This allows to review the tested functionality and test results for any given build at any given point in time.

9.4 Discussion

In this section, we discuss our findings and put them in context of related work.

Finding 1: Identified Challenges Are Partly Similar to Known Problems From Non-Agile Projects

Poor formulation of requirements and lack of traceability between artifacts are well-known problems from traditional projects [324]. Interestingly, we have also identified these two problems in our survey. The lack of traceability was observed especially for the artifacts ACC and *feature*. According to our frequency analysis, the lack of traceability between ACC was mentioned by eight of the 18 respondents, whereas the missing traceability between *features* was mentioned by only four respondents (see Figure 9.4). The challenge of inadequate formulated *acceptance criteria* was mentioned by more than 80 % of the respondents and therefore has the highest frequency in our sample. Our results indicate

that already known problems from traditional software projects could not be solved by the shift to ASD. Regardless of the development paradigm applied, practitioners seem to face the same quality issues in some of their used test artifacts.

Implications for Practice and Academia:

If teams switch to an agile paradigm, common requirements and test engineering problems still need explicit attention and won't go away by themselves. Academia will continue evaluating whether solutions applied in traditional software engineering, also work in ASD.

Finding 2: Quality Model Contains Only Currently Relevant Quality Factors

There already are a number of studies on quality factors of artefacts used in traditional projects. For example, it exists a large body of work on quality dimensions of data [325, 326, 327]. An integrated view is provided by Catarci and Scannapieco [328], who define the quality of data by the criteria accuracy, completeness, consistency, and timeliness. We were surprised that the interviewed practitioners mentioned only a few of the already known quality factors and instead emphasized specific factors. For example, in the context of *test data*, the practitioners only mentioned boundary values that the *test data* must cover. Completeness of *test data*, i.e. *"the degree to which a given data collection includes data describing the corresponding set of real-world objects"* [328], seems to be a critical problem in agile testing. The other quality factors were not discussed. We assume that the practitioners have less pressing problems in maintaining these factors and therefore do not mention them explicitly. Our problem-oriented question approach focuses on current and critical problems. As a result, our quality model contains only those quality factors that are difficult to achieve by the practitioners. This can be seen as a strength, but also as a weakness, since it will produce an incomplete model, yet provide a quality model of what is most relevant. Since project pressure is such a dominant topic in our interviews, we would argue that this makes the model more useful for practitioners.

Implications for Practice and Academia:

If practitioners do not have enough time for full-blown *Quality Assurance* (QA) of test artifacts, we suggest to start with the quality factors mentioned by fellow practitioners. Academia however, needs to validate the quality model in particular regarding relative relevance of the quality factors.

Finding 3: Most Identified Quality Factors Cannot Be Controlled Manually

Multiple studies have shown that the high change dynamics and development speed in agile projects requires an increasing automation of the test process. For example, one of our previous studies [1] stresses the need for an automatic *test case* derivation from *acceptance criteria*. Our survey indicates that quality control of test artifacts should also be automated as far as possible. We identified a number of quality factors which should be controlled since they have a significant impact on testing activities. However, they cannot be managed manually due to time constraints. The interviewed practitioners are

aware of many of the identified quality factors, but cannot meet them without automated tool support. The need for tool support primarily concerns all quality factors related to traceability such as QF 4, 5, 6, and 10.

Implications for Practice and Academia:

Academia and practice need to collaborate on creating effective and efficient tool support for automatic quality control of test artifacts.

Finding 4: The Quality of Test Artifacts Influences the Quality of Other Artifacts Indirectly

Based on the quality-in-use paradigm, we reported which quality factors of a test artifact have a positive or negative impact on certain activities. Our interviews revealed, however, that the presence or absence of the identified quality factors not only affects the activity itself but also its output and thus another test artifact which is used in subsequent activities. This can be illustrated by the two artifacts ACC and *acceptance test*. We identified seven quality factors of a ACC, which have an impact on *acceptance test design*. The output of this activity are *acceptance tests* (see Figure 9.3). Consequently, the quality of *acceptance tests* is indirectly impacted by the quality of ACC as they influence the activity in which *acceptance tests* are created. This reflects the common claim that quality defects in early artifacts (e.g., requirements-like artifacts) have consequences across multiple layers of indirection.

Implications for Practice and Academia:

Practitioners should carefully analyze which artifacts are at the beginning of the processes and focus their limited QA resources on these artifacts, in particular on ACC. Academia should try to understand this in more depth and further qualify and quantify the impact.

9.5 Threats to Validity

As in every empirical study, our survey is also subject to potential validity threats. This section discusses these threats, and describes how we mitigated them. We classify the threats into internal, construct, external, and conclusion validity [287].

Internal Validity The interviewees may have misunderstood the questions resulting in poor quality or invalid answers. To minimize this threat, we followed the guidelines by Ciolkowski [285] in the creation of the questionnaire. In addition, we conducted a pilot phase to validate the questionnaire internally through discussions in the research team and externally through pilot interviews. Another threat is that the interviewed practitioners may not have the necessary knowledge to provide suitable input to our study. We minimized this threat by selecting practitioners based on previously defined criteria to ensure sufficient experience. As in every interview-based survey, practitioners' statements may be incorrect due to fear, pride or other subjective biases, despite us

stressing the anonymity of the study. As such, our resulting quality model reflects the subjective views on quality and needs to be validated with experiments. The selection bias is another threat to internal validity. Although we have started with personal contacts to find participants, the sampling process has been extended by indirect contacts (snowball sampling). As a result, the selection bias threat has been reduced. Our study is also subject to a potential researcher bias, because all interviews and the data analysis were conducted only by the first author. To minimize this threat, all interviews were audio recorded to document the results of the interviews and to provide a basis for further analysis. In addition, the hypotheses and quality factors derived by the first author were validated by an internal review process in order to mitigate confirmation bias. Furthermore, we assured credibility by sending the identified quality factors to the participants for validation (member checking).

Construct Validity The questionnaire might not sufficiently cover our research questions limiting the availability of data that provides suitable answers to the research questions. To minimize this threat, we performed two mitigation actions. First, we designed the questionnaire to successively identify the individual elements of the ABAQM. In addition, we mapped the questions of the questionnaire to the research questions and discussed in the research group if the questions are adequate or if further questions are required to answer the RQ in a targeted way.

Reliability As in every interview-based survey, the limited sample size and the sampling strategy do not provide the statistical basis to generalize the results of the study beyond the studied companies and stakeholders. However, we tried to interview practitioners in different roles from different domains and companies of different sizes to obtain a comprehensive picture of the quality of the test artifacts. Nevertheless, the results of our frequency analysis are not statistically representative and do not allow a general conclusion about challenges in using test artifacts. In order to achieve reasonable generalizabilty, future studies should investigate our derived hypotheses in the context of a broader survey and assess their relevance, e.g. by using a Likert scale.

9.6 Summary

Quality of test artifacts matters. In this chapter, we conducted an industrial survey to create an *Activity-Based Artifact Quality Model* to define what this means from a stakeholder's viewpoint. Specifically, we explored quality factors of test artifacts that have a positive or negative impact on the activities of agile testers. Our quality model contains 16 quality factors for six test artifacts that are reportedly relevant to at least five stakeholders in the process. We encourage agile testers to use our quality model as the foundation for systematic quality control in practice.

One of the most frequently stated problems concerned the lack of adequate acceptance tests (see Figure 9.4). We found that acceptance tests are often not systematically created, resulting in incomplete or excessive test cases. In the case of missing test cases, system defects are not (or only partially) detected. In contrast, excessive test cases lead to unnecessary testing efforts and increased test maintenance costs. Consequently, practitioners need to strike a balance between full test coverage and number of required

test cases. We also found that the creation of acceptance tests is a predominantly manual task due to insufficient tool support. These results emphasize the need for a tool-supported approach capable of deriving the minimal set of required test cases automatically from NL requirements.

Focus of This Thesis In this thesis, we focus on the problem of creating suitable acceptance tests and demonstrate how conditional extraction can be used to address this problem. Specifically, we deal with the two quality factors **positive and negative scenarios** (QF 7) and **minimal** (QF 8) by proposing an approach capable of automatically generating acceptance tests in accordance with these two quality factors (see Chapter 10). The focus of this thesis is highlighted in Figure 9.3.

Chapter 10

Empirical Study on Utilizing CiRA for Automatic Acceptance Test Creation

Common Thread In the previous Chapter 9, we provided empirical evidence that practitioners fail to create adequate acceptance tests. In this chapter, we demonstrate how the automatic extraction of conditionals can contribute to solving this problem. Specifically, we use `CiRA` to extract conditionals in fine-grained form and then map them into a *Cause-Effect-Graph* from which we automatically derive test cases. We empirically evaluate our approach on real-world data provided by *Allianz Deutschland AG* (insurance), *Ericsson* (telecommunication), and *Leopold Kostal GmbH & Co. KG* (automotive).

Contribution Across all case companies, our approach automatically created 71.8 % of the 578 manually created test cases. Our approach was further able to identify 136 test cases that were missed in manual test design. In fact, 58.8 % of these exclusively automatically generated test cases are indeed relevant and should be included in the acceptance test. Our study proves that our approach is able to automatically create a significant amount of relevant (known and new) test cases. However, the study also shows that our approach does not achieve full automation of acceptance test generation. This is mainly due to the fact that incompleteness of requirements is still a major issue in practice making domain knowledge often necessary to create all relevant test cases. Hence, our approach should be used as a complement to the conventional manual test case derivation process.

Related Publications This chapter is taken, directly or with minor modifications, from a previous publication [10].

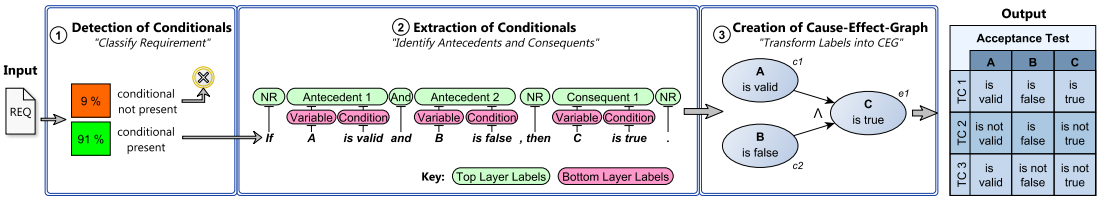


Figure 10.1: Overview of Our Approach Consisting of Three Steps: (1) Detection of Conditionals, (2) Fine-Grained Extraction of Conditionals, and (3) CEG Creation. Processed REQ: *"If A is valid and B is false, then C is true."*

10.1 Combining CiRA With Cause-Effect-Graphing

This section highlights our idea of using conditional extraction for automated acceptance test creation. First, we describe why and how we combine our CiRA pipeline with *Cause-Effect-Graphing* to generate test cases automatically (see Section 10.1.1 and Section 10.1.2). Second, we demonstrate our approach by means of a running example (see Section 10.1.3).

10.1.1 Principal Idea

We follow the idea of *Model-Based-Testing* and introduce an intermediate layer between requirements and corresponding acceptance tests. Specifically, our approach consists of three steps: First, we use CiRA to extract conditionals from the requirements. Second, we transfer them into a test model from which we then can derive test cases automatically. We choose *Cause-Effect-Graphs* as suitable test models for the following reasons:

1. Since we focus on conditional statements, we need to determine the right combination of test cases that cover the relationship between the extracted *antecedents* and *consequents*. *Cause-Effect-Graphs* are ideally suited for this purpose because they can be interpreted as a combinatorial logic network, which describes the interaction of *antecedents* and *consequents* by Boolean logic. Specifically, it consists of nodes for each *antecedent* and *consequent* and uses arcs with Boolean operators (conjunction $\wedge$, disjunction $\vee$, negation $\neg$) to illustrate the relationship between the nodes (see Section 2.4).

2. To derive test cases from a *Cause-Effect-Graphs*, we apply the *Basic Path Sensitization Technique*. The graph is traversed back from the *consequents* to the *antecedents* and test cases are created according to specific decision rules (see Section 2.4). These rules achieve the maximum probability of finding failures while avoiding the complexity of generating 2^n test cases, where n is the number of *antecedents*. Hence, *Cause-Effect-Graphing* allows us to support practitioners in balancing between sufficient test coverage and the lowest possible number of test cases.

10.1.2 Creation of Cause-Effect-Graph

As shown in Figure 10.1, we produce a CEG based on the extracted *antecedents* and *consequents* by CiRA. Creating a CEG is not a trivial task, especially for complex con-

ditional statements consisting of multiple *antecedents* and *consequents*. We handle the following cases:

Case 1: Single Antecedent - Single Consequent In the simplest case, we create two nodes and draw an edge from the *antecedent* node to the *consequent* node.

Case 2: Multiple Conjunctive Antecedents - Single Consequent In this case, all *antecedents* must occur jointly for the effect to occur. Thus, we connect all *antecedent* nodes with the *consequent* node using the connective $\wedge$.

Case 3: Multiple Disjunctive Antecedents - Single Consequent In this case, the occurrence of one of the *antecedents* is sufficient for the *consequent* to occur. Thus, we link all *antecedent* nodes with the *consequent* node using the logical connective $\vee$.

Case 4: Combination of Conjunctive and Disjunctive Antecedents - Single Consequent Conditionals are usually not parenthesized, which causes a certain degree of ambiguity when combining conjunctions and disjunctions. To convert such conditionals uniformly into a CEG, we follow the precedence rules of *Propositional Logic*. Hence, we evaluate conjunctions with higher precedence than disjunctions. To this end, we create an intermediate node for each set of conjunctive *antecedents* and connect the *antecedents* with the intermediate node using the logical connective $\wedge$. Subsequently, we connect the disjunctive *antecedent(s)* and the intermediate node(s) with the *consequent* using the logical connective $\vee$. If no connective can be found between two adjacent *antecedents*, the closest subsequent connective is used. For example, in an enumeration like *"Owners, tenants, and managers"* only the connection between *"tenants"* and *"managers"* is explicit, whereas *"owners"* and *"tenants"* are also implicitly connected by a conjunction.

Case 5: Multiple Conjunctive Consequents We create a node for each *consequent* and connect them to the *antecedents* according to the rules described above. We do not allow effects to be connected with a disjunction as this would denote an indeterministic system behavior.

Case 6: Correction of Incomplete nodes In the simplest case, a *antecedent* or *consequent* encompasses both a variable and condition in the lower annotation level. We then fill the created nodes with the corresponding information. If either of the two labels is missing, the information is extracted from the nearest referent instead.

10.1.3 Motivating Example

We demonstrate the functionality of our pipeline by means of a running example. Specifically, we explain how we automatically derives the minimum number of required test cases for the requirements specification shown in Figure 10.2. The specification contains an excerpt of requirements that describe the functionality of *The Energy Management System* (THEMAS). THEMAS is intended to be used by people that maintain the heating and cooling systems in a building. We retrieved the requirements from the PURE data set [218] that contains 79 publicly available NL requirements documents collected

- **REQ A:** If the temperature change is requested, then the determine heating/cooling mode process is activated and makes a heating/cooling request.
- **REQ B:** If the current temperature value is strictly less than the lower value of the valid temperature range or if the received temperature value is strictly greater than the upper value of the valid temperature range, then the THEMAS system shall identify the current temperature value as an invalid temperature and shall output an invalid temperature status.
- **REQ C:** The THEMAS system shall maintain the ON/OFF status of each heating and cooling unit.
- **REQ D:** Temperatures that do not exceed these limits shall be output for subsequent processing.
- **REQ E:** If this condition is true, then this module shall output a request to turn on the heating unit in case LO = T LT.
- **REQ F:** The heating/cooling unit shall have no real time delay when these statuses are sent to the THEMAS system.
- **REQ G:** Each thermostat shall have a unique identifier by which that thermostat is identified in the THEMAS system.
- **REQ H:** When an event occurs, the THEMAS system shall identify the event type and format an appropriate event message.

Figure 10.2: Requirements Specification of THEMAS, Based on [218].

from the Web. We encourage the readers of this chapter to use our online demo to process the running example on their own, allowing them to follow each individual step of our test case generation pipeline.

Step 1: Detection of Conditionals When applying CiRA to the requirements specification shown in Figure 10.2, CiRA performs the first step of its pipeline: it tries to identify which requirements in the specification contain a conditional. For this purpose, REQ A - H are tokenized in the embedding layer and enriched with dependency tags as described in the previous section. Finally, the CLS token of each REQ is passed to the inference layer, where a softmax layer computes the probability of the REQ being a conditional or not. In the present example, CiRA classifies REQ A, REQ B, REQ D, REQ E, REQ F, and REQ H as Conditional Present and correctly discovers that REQ C and REQ G do not contain a conditional statement. Hence, REQ C and REQ G are excluded from the further test generation process. The remaining requirements are forwarded to the next step, namely the extraction of conditionals.

Step 2: Extraction of Conditionals In the second step, CiRA extracts the conditional statements from the requirements that were classified as Conditional Present in the first step. For this purpose, REQ A, REQ B, REQ D, REQ E, REQ F, and REQ H are decomposed into individual tokens using the BPE tokenizer and then converted into RoBERTa embeddings. Subsequently, each token embedding is fed into a sigmoid classifier, which calculates the probability for each of our twelve labels (Antecedent 1,

Antecedent 2, ... Condition) that the token should be associated with that class. We select the classes with a probability ≥ 0.5 as the final classification result. Figure 10.3 shows the extracted conditionals by CiRA from the THEMAS requirements.

The running example demonstrates that CiRA is able to extract conditionals in fine grained form - independent of whether the requirements contain simple conditionals consisting of a single *antecedent* and *consequent* (see REQ D and REQ F) or complex conditionals with multiple *antecedents* and *consequents* (see REQ B). Further, CiRA is able to detect *antecedents* and *consequents* in different positions in a sentence, which can be illustrated by REQ E. In this case, CiRA extracts the conditional statement correctly even though Antecedent 1 and Antecedent 2 do not immediately follow each other but instead are separated by Consequent 1.

Step 3: Creation of Cause Effect Graph In the third step, we interpret the conditional statements extracted from REQ A, REQ B, REQ D, REQ E, REQ F, and REQ H, and create a corresponding CEG. Subsequently, we apply the BPST (cf. [148] and [64]) to derive the minimum number of required test cases from each CEG. Figure 10.3 presents an overview of the *Cause-Effect-Graphs* created by our approach and the respective automatically generated test specifications for each conditional included in the THEMAS requirements specification (see Figure 10.2).

The CEG generated for REQ A corresponds to **Case 5** described in Section 10.1.2. Specifically, one *antecedent* is the trigger for two conjunctive causes: *"the heating/cooling process is activated"* and *"[the heating/cooling process] makes a heating/cooling request"*. We can observe from the extracted conditional for REQ A that the variable of Consequent 2 is not explicitly defined in the requirement (see Figure 10.3a). We must therefore automatically complete the node of Consequent 2 by adopting the variable from Consequent 1 (see **Case 6**). REQ B specifies complex system behavior and contains two *antecedents* and two *consequents*. When creating a corresponding CEG, we need to consider both **Case 2** and **Case 5**. Thus, we link all *antecedent* nodes with both *consequent* nodes using the logical connective ∨ (see Figure 10.3b). Similar to REQ A, we need to automatically complete the variable of Consequent 2 since it is not explicitly defined in REQ B (see **Case 6**). REQ D contains a negated *antecedent* that is responsible for the occurrence of a single *consequent*. In other words, not exceeding the limit is required for a temperature to be eligible for further processing. Therefore, we are dealing with **Case 1** and have to negate the edge between *antecedent* and *consequent*. Once again, the variable of an *consequent* node is not described in the requirement. In contrast to REQ A and REQ B we do not complement the node of Consequent 1 with the variable of a neighboring *consequent* but rather adopt the variable of Antecedent 1 (see Figure 10.3c). The CEG generated for REQ E corresponds to **Case 2**. Hence, we connect both *antecedent* nodes with the single *consequent* using the connective ∧ (see Figure 10.3d). REQ F includes a negated *consequent* that is triggered by a single *antecedent*. We thus create the CEG based on the rules described in **Case 1** and negate the edge between *antecedent* and *consequent* (see Figure 10.3e). REQ H contains one *antecedent* and two conjunctive *consequents*. We create a corresponding CEG by applying the rules described in **Case 5**. We complement the node of Consequent 2 by adopting the variable of Consequent 1 (see Figure 10.3f).

Our approach automatically created a total of 14 test cases for all conditionals included in our running example. The created acceptance tests for REQ D and REQ F are

trivial and contain only two parameters that have to be checked (see Figure 10.3c and Figure 10.3e). The other acceptance tests are of higher complexity as they contain more input and output parameters: To fully test REQ A, REQ E and REQ H, two test cases including three parameters each have to be checked. The acceptance test created for REQ B involves three test cases with four parameters.

10.2 Research Objective

We aim to investigate whether our approach is suitable for the automatic generation of acceptance tests in practice. Specifically, we study the following research questions (RQ):

- **RQ 1:** Can our automated approach create the same test cases as the manual approach?
- **RQ 2:** What are the reasons for deviating test cases?

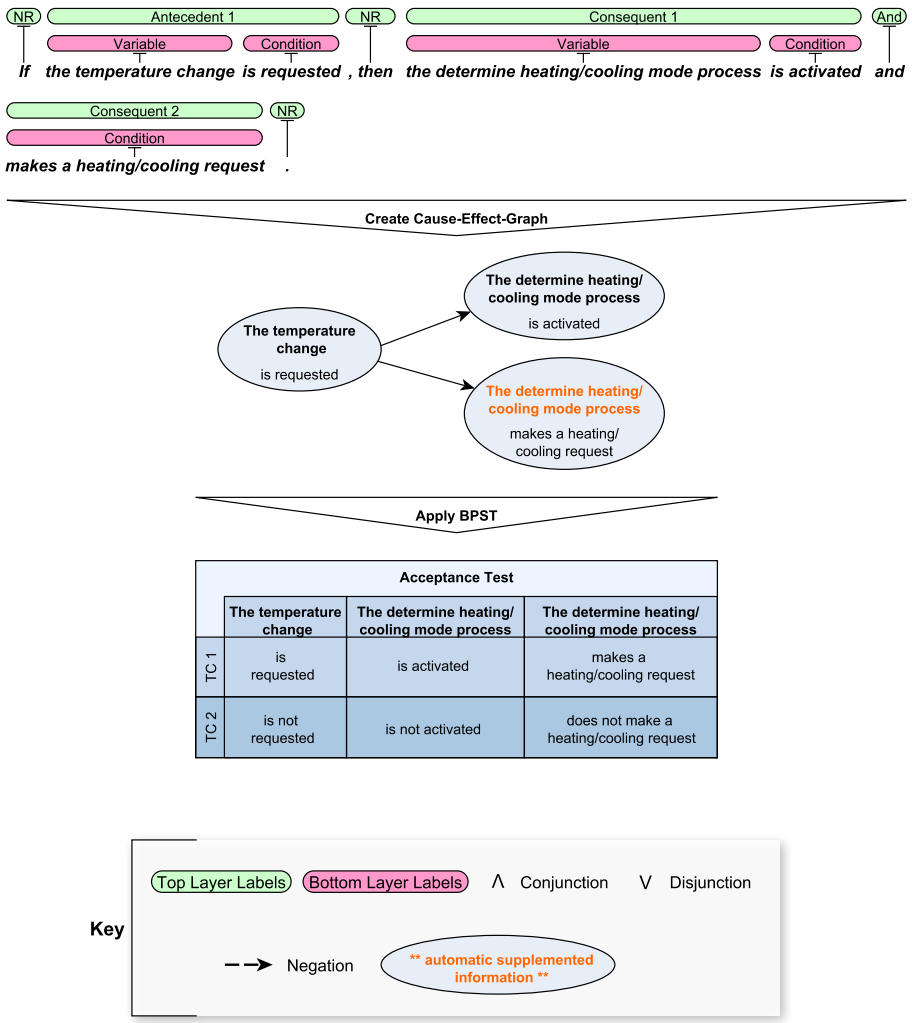
RQ 1 and RQ 2 inspect the impact of our approach: does it achieve the status quo or even lead to an improvement of the manual test case derivation? To this end, we conduct a case study with three industry partners in an exploratory fashion and compare automatically created test cases with existing, manually created test cases. For our study, we follow the guidelines by Runeson and Höst [269] for conducting case study research.

10.3 Study Design

Case Sampling and Study Objects We apply purposive case sampling augmented with convenience sampling [329]. Specifically, we approached some of our industry contacts inquiring whether they are interested in exploring the potential of `CiRA`. We were provided with data from three companies operating in different domains: *Allianz Deutschland AG* (insurance), *Ericsson* (telecommunication), and *Leopold Kostal GmbH & Co. KG* (automotive). Since the data is subject to non-disclosure agreements, we are unable to share the provided requirements and test cases.

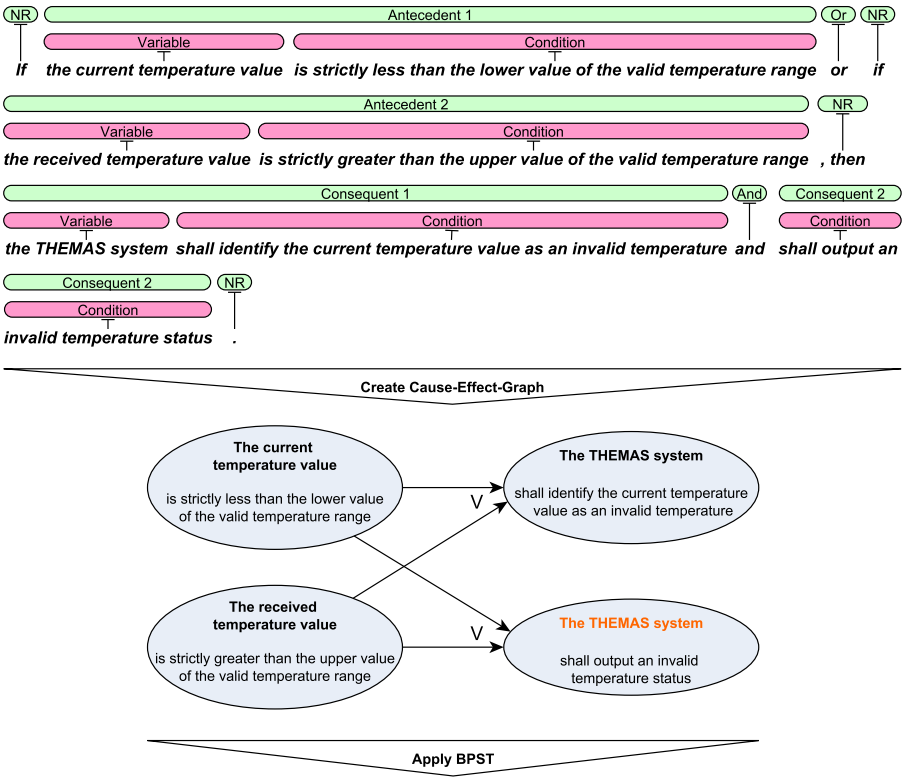
Allianz Data We analyze 219 ACC describing the functionality of a business information system used for vehicle insurance. 127 of these ACC contain conditionals and are therefore suitable for assessing `CiRA`. The remaining ACC specify the expected functionality based on process flows (16 criteria) or in a static way (76 criteria). We analyze the acceptance tests that were manually created for each of the ACC including conditionals. In total, 309 test cases were designed, which corresponds to about 2.43 test cases per acceptance test.

Ericsson Data We analyze 109 requirements derived from five *Business Use Cases* (BUCs), which are feature-level units of development at *Ericsson*. The BUCs originate from different functional topics. 49 of these 109 requirements contain conditionals while the remaining requirements are expressed in a static way. In total, 65 test cases were manually generated for the 49 requirements containing conditionals, which corresponds to about 1.33 test cases per acceptance test.



(a) Extracted Conditional From REQ A, Corresponding CEG, and Derived Acceptance Test.

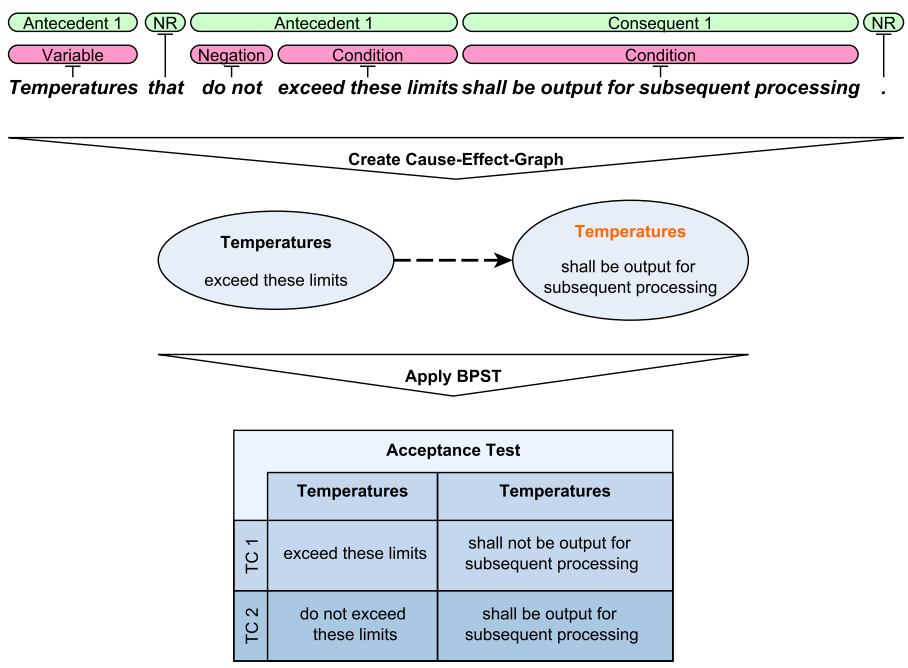
Figure 10.3: Overview of the Conditionals Extracted by CiRA in Fine-Grained Form, the Generated *Cause-Effect-Graphs*, and the Derived Acceptance Tests per Requirement Defined in the THEMAS Specification.



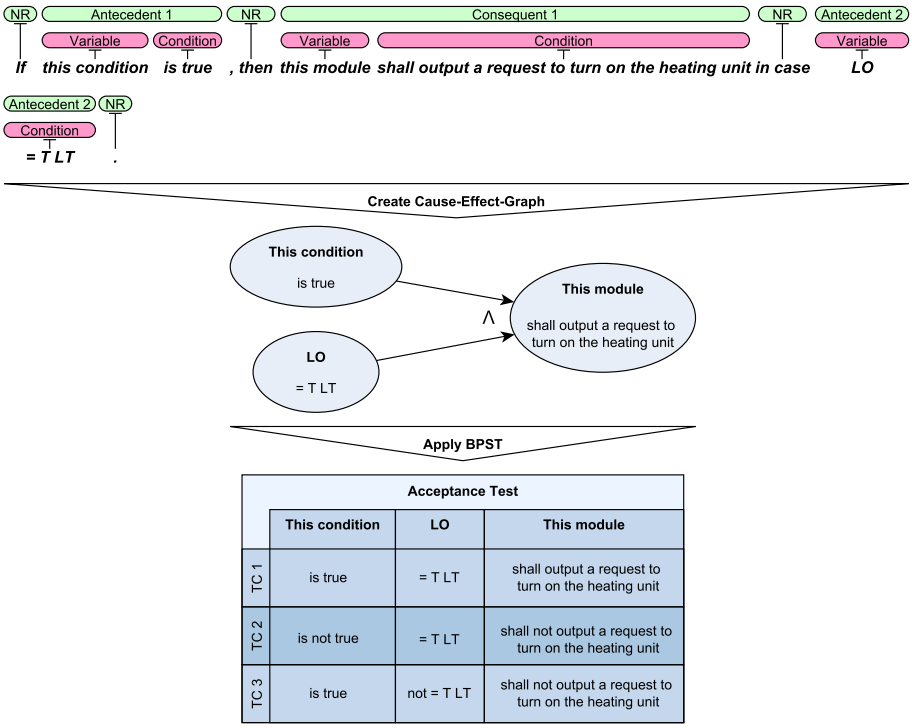
Acceptance Test				
	The current temperature value	**The received temperature value**	**The THEMAS system**	**The THEMAS system**
TC 1	is strictly less than the lower value of the valid temperature range	is not strictly greater than the upper value of the valid temperature range	shall identify the current temperature value as an invalid temperature	shall output an invalid temperature status
TC 2	is not strictly less than the lower value of the valid temperature range	is strictly greater than the upper value of the valid temperature range	shall identify the current temperature value as an invalid temperature	shall output an invalid temperature status
TC 3	is not strictly less than the lower value of the valid temperature range	is not strictly greater than the upper value of the valid temperature range	shall not identify the current temperature value as an invalid temperature	shall not output an invalid temperature status

(b) Extracted Conditional From REQ B, Corresponding CEG, and Derived Acceptance Test.

Figure 10.3: Overview of Automatically Generated Acceptance Tests (cont.)

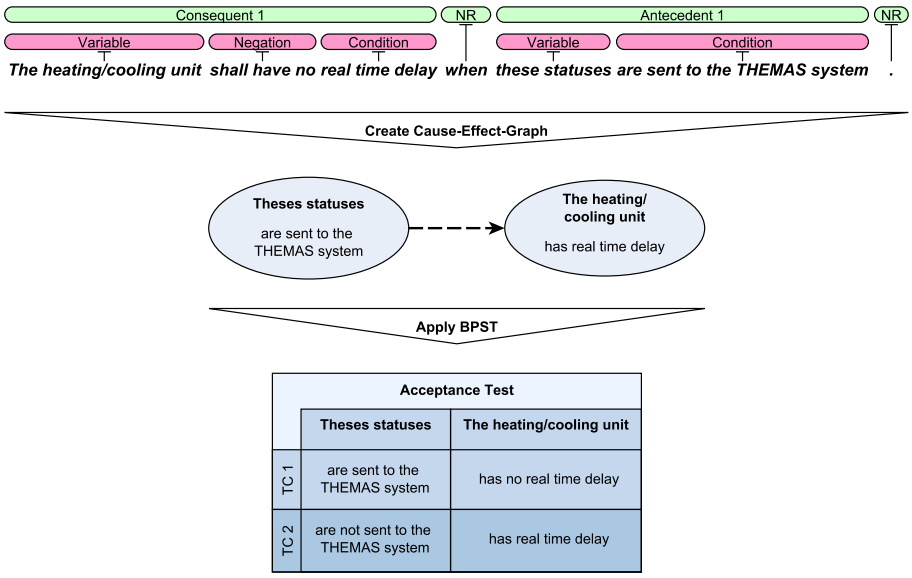


(c) Extracted Conditional From REQ D, Corresponding CEG, and Derived Acceptance Test.

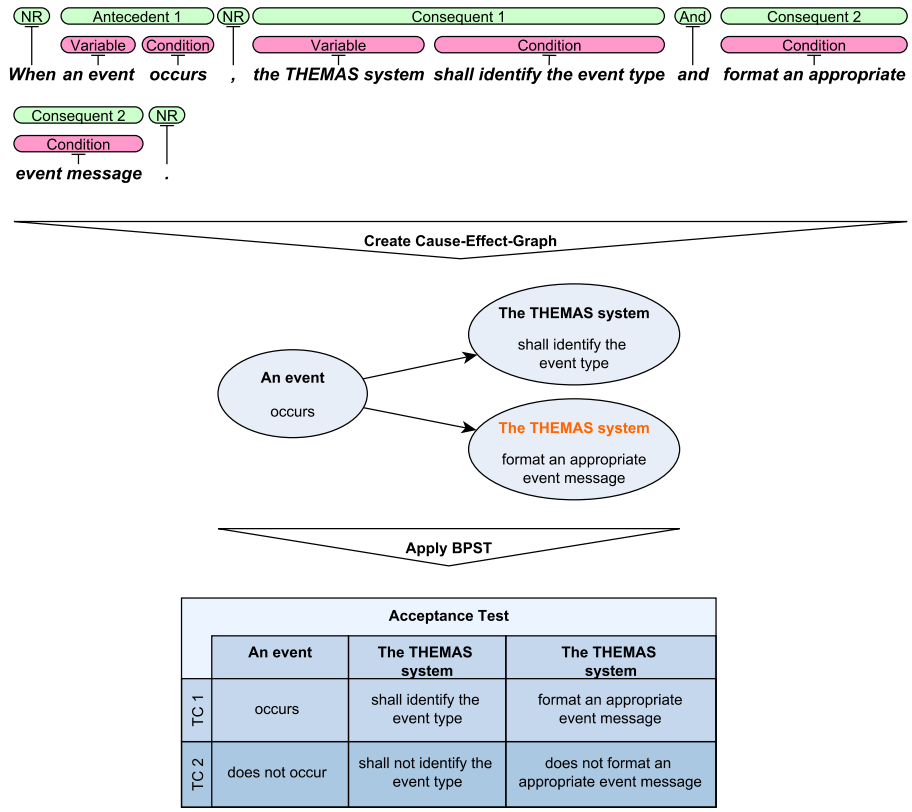


(d) Extracted Conditional From REQ E, Corresponding CEG, and Derived Acceptance Test.

Figure 10.3: Overview of Automatically Generated Acceptance Tests (cont.)



(e) Extracted Conditional From REQ F, Corresponding CEG, and Derived Acceptance Test.



(f) Extracted Conditional From REQ H, Corresponding CEG, and Derived Acceptance Test.

Figure 10.3: Overview of Automatically Generated Acceptance Tests (cont.)

Kostal Data We analyze a requirements specification describing a plug interlock function, which prevents a charging plug from being disconnected during an active charging process of an electric car. The specification includes 135 functional requirements. 79 of these functional requirements contain conditionals while 56 requirements describe the functional behavior in a static way: *"The signal* `signalName` *shall be set to* `InitValue`*"*. In our case study, we focus only on the acceptance tests that were manually created for the 79 requirements that contain conditionals. In total, 204 test cases were designed, which corresponds to about 2.58 test cases per acceptance test.

Approach for RQ 1 We want to study whether our approach can achieve the status quo or even lead to an improvement of the manual test case derivation. To this end, we pass all study objects through our pipeline and compare the automatically created acceptance tests with the manually created acceptance tests. We assess two acceptance tests to be equal if they contain the same test cases. Two test cases are equivalent if they consist of the same input and output parameters with semantically identical variables and conditions. However, we allow syntactical differences between the test cases (e.g., different spelling of parameters), since they still test the same functionality. By comparing the test cases created by our approach with the manually created test cases, we found that it is sometimes not possible to establish a one-to-one relationship. Partly, test designers aggregate related parameters, so that a manual test case may cover multiple automated test cases (one-to-many relationship). Therefore, two acceptance tests may also be equivalent even if the number of test cases differs. If we observe discrepancies between a manual acceptance test and automatic acceptance test, we involve test designers from our case companies and examine the set differences: 1) test cases created exclusively by the manual approach (MA), and 2) test cases generated exclusively by our automated approach (AA). In both cases, we ask the test designers whether a certain test case is required to fully check the functionality described by the requirement to assess its *relevance* (rel). Consequently, we investigate five different categories of test cases:

- $\boldsymbol{Identical}$: A test case that has been created manually as well as automatically by our approach.
- $\boldsymbol{AA \wedge rel}$: A test case that has been missed in manual test design and should be included in the acceptance test.
- $\boldsymbol{AA \wedge \neg rel}$: A superfluous test case that is correctly not included in the manually created acceptance test.
- $\boldsymbol{MA \wedge rel}$: A test case that has been missed by our approach and should be included in the acceptance test.
- $\boldsymbol{MA \wedge \neg rel}$: A superfluous test case that is correctly not included in the automatically created acceptance test.

Approach for RQ 2 To answer the second RQ, we document errors of our approach and interview test designers of our case companies. To avoid interviewer bias, we do not involve the test designers who created the respective acceptance tests. Instead, we

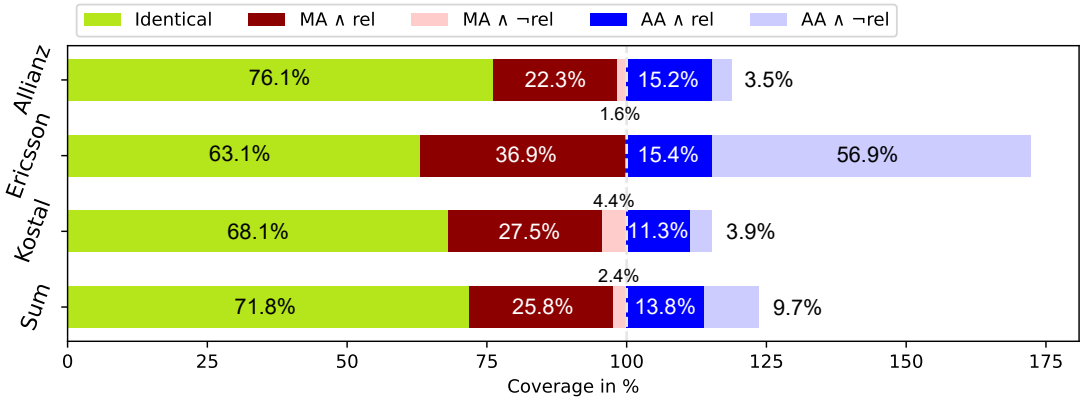


Figure 10.4: Case Study Results. Comparison of Manually and Automatically Created Test Cases.

interview their colleagues who are also familiar with the functionalities described in the requirements. To determine the reasons for deviating acceptance tests, we examine the manually and automatically created test case as well as the corresponding requirements jointly with the test designers. We involve two test designers at *Allianz*, two test designers at *Kostal*, and one test designer at *Ericsson*.

10.4 Study Results

In this section, we report on our study results structured by our research questions.

RQ 1: Can CiRA Create the Same Test Cases as the Manual Approach?

Findings at Allianz CiRA detected 90.55 % of the conditionals in the acceptance criteria. Consequently, no test cases were created for the missed 12 criteria containing conditionals. For the correctly classified criteria, our approach generated 314 test cases. This corresponds to about 2.73 test cases per acceptance test. We were able to draw a one-to-one relationship between 224 manually and automatically created test cases. Additionally, we observed a one-to-many relationship between eleven manually created test cases and 32 automatically created test cases. Thus, 76.05 % of the manually created test cases could be automatically generated. However, 74 test cases were not created by our approach, of which 27 test cases are related to criteria that were incorrectly identified as Conditional Not Present. According to the test designers, the remaining 47 MA test cases can be classified as follows: 42 are necessary to fully test the system functionality while five test cases are superfluous. A comparison of the automatically created test cases with the manually created test cases highlights that 58 test cases have not yet been considered in the manual test design. According to the test designer, these 58 AA test cases can be clustered as follows: 47 are indeed *relevant* while eleven should not be included in the acceptance test.

Findings at Ericsson CiRA correctly classified 79.6 % of the conditionals in requirements but failed to do so for ten requirements. 91 test cases were automatically gener-

ated based on these identified requirements, which corresponds to about 2.33 test cases per acceptance test. 28 manual test cases were automatically created by our approach in a one-to-one, 13 more in a one-to-many relationship, resulting in an automatic generation of 41 of 65 test cases (63.1 %). However, 24 test cases were not created by our approach, of which seven test cases are related to criteria that were incorrectly identified as Conditional Not Present. According to the test designer, the remaining 17 MA test cases are all necessary to fully test the system's functionality. A comparison of the automatically created test cases with the manually created test cases highlights that 47 test cases have not yet been considered in the manual test design. According to the test designer, these 47 AA test cases can be clustered as follows: ten are indeed *relevant* while 37 should not be included in the acceptance test.

Findings at Kostal CiRA correctly classified 72 requirements as Conditional Present. However, it failed to identify the remaining seven requirements that also contain conditionals. Hence, no test cases were ultimately created for these requirements. In the case of the correctly classified requirements, CiRA produced 194 test cases. This corresponds to about 2.69 test cases per acceptance test. We found a one-to-one relationship between 122 manually and automatically created test cases. In addition, we were able to draw a one-to-many relationship between 17 manual test cases and 41 automatically created test cases. Thus, 68.14 % of the manually created test cases could be created automatically. Nevertheless, 65 manually created test cases are not included in the set of automated test cases. 16 of these exclusively manually created test cases refer to the conditionals in requirements that CiRA missed. In the case of the other 49 test cases, we ask test designers at *Kostal* about their relevance. In fact, 81.63 % of the exclusively manually created test cases are deemed *relevant*. According to the test designers, nine test cases are superfluous and can be removed from the test set. Examining the automatically created test cases, we observe that 31 test cases have not been considered in the manual creation so far. Interestingly, the test designers confirmed that 74.19 % of these test cases were indeed missed in the manual process. However, eight exclusively automatically created test cases are not *relevant* and thus correctly not included in the manual set.

Answer to RQ 1:

Across all case companies, our approach automatically created 71.8 % of the 578 manually created test cases. Our approach was further able to identify 136 test cases that were missed in manual test design. In fact, 58.8 % of these exclusively automatically generated test cases are indeed relevant and should be included in the acceptance test. We conclude that our approach is able to automatically create a significant amount of relevant (known and new) test cases.

RQ 2: What Are the Reasons for Deviating Test Cases?

Incomplete Requirements We found that the main reason for test cases that could not be created automatically lies in the poor information available in the requirements. The interviewed test designers confirmed that domain knowledge is often required to determine all relevant test cases. In the case of *Kostal*, 19 out of 79 requirements were incomplete. We found that our approach could not generate 37 $\boldsymbol{MA} \wedge \boldsymbol{rel}$ test cases due to lack of

information in these requirements. At *Allianz*, 16 out of 127 conditionals in acceptance criteria lack information. Our analysis shows that our approach could not generate 31 $\boldsymbol{MA} \wedge \boldsymbol{rel}$ test cases due to incomplete acceptance criteria. At *Ericsson*, 17 $\boldsymbol{MA} \wedge \boldsymbol{rel}$ test cases could not be generated due to underspecified or missing requirements.

Incorrect Combinatorics We noticed that some of the exclusively manually created test cases are superfluous - they can be merged or are already covered by other test cases. The interviews revealed that in these cases the combinatorics of the input and output parameters were interpreted incorrectly. According to the test designers, this stems mainly from the fact that test cases are often not created systematically, but rather based on past experience. Unsystematic test design may not only result in superfluous test cases but can also lead to necessary test cases being ignored. We observed that test designers tend to create positive cases and neglect negative cases. At *Kostal*, 21 of the 23 $\boldsymbol{AA} \wedge \boldsymbol{rel}$ test cases were actually negative cases. Only two positive cases were overlooked in the manual process. At *Allianz*, 36 of the 47 $\boldsymbol{AA} \wedge \boldsymbol{rel}$ test cases were actually negative cases. 11 positive cases were missed by the test designers. In the case of *Ericsson*, all ten $\boldsymbol{AA} \wedge \boldsymbol{rel}$ test cases were overlooked negative test cases.

Infeasible Test Cases Our analysis shows that some of the exclusively automatically created test cases can not occur in practice. According to the test designers, this problem arises mainly for negative test cases where certain scenarios are tested that can only occur theoretically. For example, some parameters can not take the value false at the same time, even if this case should be checked from a combinatorial point of view. In the case of *Kostal*, we found that three of the eight $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases can not be checked in practice. At *Allianz*, five of the eleven $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases can only occur theoretically. At *Ericsson*, 28 of 37 $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases fell into this category.

Errors in Our Pipeline Our approach produced not only errors in the detection of the conditionals, but also failed in some cases to extract and translate them into the CEG. At *Kostal*, our approach failed to generate 3 $\boldsymbol{MA} \wedge \boldsymbol{rel}$ test cases and instead created five $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases, because the generated CEG reflected a wrong conditional statement. In the case of *Allianz*, we failed to create eleven $\boldsymbol{MA} \wedge \boldsymbol{rel}$ test cases and instead generated six $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases. In the case of *Ericsson*, our approach produced nine $\boldsymbol{AA} \wedge \boldsymbol{\neg rel}$ test cases due to incorrect interpretation of the conditional. We found that these errors occurred mainly when the conditionals contained three or more *consequents*. This confirms the findings from our experiment that `CiRA` struggles in reliably identifying more than two *consequents* (see Table 7.6).

Answer to RQ 2:

In our setting, we observed four reasons for deviating test cases: incomplete requirements, incorrect combinatorics, infeasible test cases, and errors in our pipeline. We found that incomplete requirements are the main reason for test cases that could not be created automatically by our approach.

10.5 Discussion

Our case study demonstrates that our approach is able to support practitioners in deriving relevant test cases from conditionals. Across all industry partners, our approach automatically generates more than 70 % of the manually created test cases. However, our approach does not achieve full automation of acceptance test creation, mainly due to incomplete requirements. Our approach is heavily dependent on the information contained in the requirements and consequently unable to create test cases for which additional domain knowledge is required. Thus, our case study confirms the findings of Mendez et al. [44] that incompleteness is still a major problem in practice and hinders the automatic processing of requirements.

1. Key Take-away:

In fact, our approach can help to generate acceptance tests automatically. However, our approach does not substitute a test designer since domain knowledge is often necessary to identify all required test cases.

According to the test designers, the main benefit of our approach is its ability to create test cases automatically based on heuristics. Hence, it is independent of human bias and able to identify test cases that may be missed in the manual process. We argue that our approach should always be used as a supplement to the existing manual process to highlight all test cases that should be tested from a combinatorial point of view, in particular negative test cases that were proportionally more often overlooked than positive test cases. The automatically generated set of test cases may then be manually extended by test cases that require domain knowledge. At *Ericsson*, we observed that a large amount of automatically generated test cases were irrelevant since they can only occur theoretically. Hence, when utilizing our approach as a supplement to manual test design, test designers need to filter the automatically generated test cases. However, we argue that it is easier to discard infeasible test cases than to manually identify undetected relevant test cases.

2. Key Take-away:

Our approach is particularly useful for automatically identifying negative test cases, which are often overlooked in the manual creation process. However, not all test cases created by our approach are necessarily relevant, requiring subsequent manual review of the automatically created test specifications.

Since `CiRA` decomposes each sentence using subword tokenization and labels each token individually, it is much more robust against grammar errors and is also able to process OOV words. Nevertheless, studies [330] reveal that language models such as BERT show significant performance degradation with increasing amounts of noisy data. As a result, we hypothesize that the robustness of `CiRA` against grammatical mistakes is limited to a few errors in a sentence. We therefore propose to combine `CiRA` with requirements smell checkers [331] in the future to automatically verify the linguistic quality of requirements before passing them into the `CiRA` pipeline.

3. Key Take-away:

Fully automated acceptance test generation is difficult to achieve because requirements often suffer from poor quality. RE teams should therefore first check the quality of the requirements before processing them with CiRA.

CiRA is limited to single sentence conditionals and is not able to extract conditional statements that span multiple sentences. However, two-sentence conditionals may arise in practice (e.g., indicated by *"therefore"*, *"hence"*), requiring us to extend CiRA in future work (see Section 11.2). According to the test designers, a further challenge in the extraction of conditionals relates to the handling of *event chains* (i.e., linked requirements, in which the *consequent* of a conditional represents a *antecedent* in another conditional). In such cases, it is no longer sufficient to create a single CEG. Rather, we must create several *Cause-Effect-Graphs* and connect them to each other. Currently, CiRA only allows the creation of acceptance tests for requirements that contain conditionals. For full automation of test case design, however, we also require approaches capable of processing static requirements and process flows.

4. Key Take-away:

So far, the feasibility of CiRA is limited to conditionals that span a single sentence. As a consequence, we still need to develop methods for the automatic generation of test cases from static requirements and process flows.

Our case study focuses on a quantitative comparison between manually and automatically created test cases. However, several other metrics are available to benchmark test cases [332]. For example, structural criteria like *test understandability* investigate whether a test is easy to understand in terms of its internal and external descriptions. We plan to extend our study to obtain further insights into the quality of the test cases generated by CiRA.

10.6 Threats to Validity

Internal Validity We acknowledge a possible threat to internal validity due to selection bias of suitable requirements artifacts. In all cases, the artifact selection was driven by the availability of data. Hence, requirements and test cases were not actively sampled to improve the performance of our approach.

Construct Validity The comparison between the manually and automatically created test cases might be subject to researcher bias. To mitigate this risk, the first and second authors individually mapped the test cases. Subsequently, the mapping was cross-checked and discussed within the research group. A further threat to internal validity is the potential bias of the interviewed test designers. To keep this risk as low as possible, we interviewed each test designer independently and compared the reasons for the deviating test cases.

External Validity To achieve reasonable generalizability, we selected requirements and test cases from different domains. However, the limited sample size does not provide the statistical basis to generalize the results of our study beyond the studied case companies. Nevertheless, we hypothesize that our approach may also be valuable for other companies considering that conditionals are widely used in requirements. Validation of this claim requires further empirical investigation.

10.7 Summary

As shown in Chapter 9, the creation of acceptance tests is laborious and still requires manual work due to missing tool support. In this chapter, we demonstrate how automated conditional extraction from requirements can be used to automatically generate the acceptance tests. Specifically, we translate the extracted conditionals by `CiRA` into a CEG, from which we derive the minimal number of required test cases. We evaluate our approach by conducting a case study with three companies. Our study demonstrates that our approach is able to automatically create 71.8 % of the 578 manually created test cases. Additionally, our approach identified 80 relevant test cases that were missed in manual test design. The findings of our study prove that automated conditional extraction can indeed contribute to the implementation of Use Case 1 as described in Chapter 1. However, we do not achieve full automation of acceptance test generation mainly due to (1) incomplete requirements and (2) errors of our approach in interpreting conditionals that contain three or more *consequents*. Hence, we suggest to use our approach as a supplement to the existing manual creation process to make test designers aware of all test cases that should be tested from a combinatorial point of view. We hypothesize that this will help to significantly reduce the risk of missed negative test cases.

Chapter 11

Conclusions, Limitations, and Outlook

This chapter concludes the dissertation and describes possible directions for further research. Section 11.1 explains how we addressed the problem statement of this thesis and summarizes our main contributions. In Section 11.2, we discuss limitations of our work. In particular, we identify missing features of CiRA and indicate ways to implement them in future studies. We also highlight additional use cases to which automated conditional extraction may contribute.

11.1 Conclusions

This thesis is based on the problem statement *"we need (1) a better understanding of the notion of conditionals in requirements artifacts and (2) a comprehensive method and tool support to extract conditionals in fine-grained form"* (see Section 1.3). We claimed that this thesis addresses both problems. In the following, we conclude that this claim is supported by the *knowledge-seeking* and *solution-seeking* contributions provided in this thesis. We map our contributions to the two main questions included in the title of this thesis and explain **why and how to extract conditionals from RE artifacts**.

11.1.1 Why to Extract Conditionals From RE Artifacts?

Conditionals Are Prevalent in RE Artifacts Conditional statements (e.g., *"If A and B occur, then C evaluates to false"*) represent an intuitive means to express conditions and their consequences. In Part I, we presented two empirical studies that prove that conditionals also occur in RE artifacts. Our analysis of 14,983 sentences emerging from 53 requirement documents revealed that conditionals are included in about 28 % of the investigated sentences. Further, we studied 961 user stories describing the functionality of two BIS being in production at *Allianz Deutschland AG* (insurance) and showed that conditionals are as well prevalent in agile RE artifacts such as acceptance criteria. In fact, we found that conditionals are used in about 49 % of the investigated user stories. Hence, conditionals matter in *Requirements Engineering*, which motivates the necessity of an effective and reliable approach for the automated extraction of conditional statements from requirements.

Conditionals Contain Logical Knowledge About the Expected System Behavior Conditionals in RE artifacts specify a system from three perspectives: (1) the inputs to be processed by the system, (2) the expected system behavior once these inputs occur, and (3) the outputs that the system shall produce. Specifically, conditional statements describe the expected system behavior in terms of the interaction of certain *antecedents* and *consequents*. Automatically extracting this embedded logical knowledge can help to automate two RE tasks for which sufficient methods and tools are not yet available: *"acceptance test*

creation" (Use Case 1) and *"dependency detection between requirements"* (Use Case 2) (see Section 1.2). We analyzed the form and complexity of conditionals in RE artifacts and defined requirements for a suitable conditional extraction approach. The majority of conditionals in requirements occur in *marked* form and contain one or more cue phrases (e.g., *"if"*, *"then"*) to indicate the dependency between certain events. The complexity of conditionals is confined since they usually consist of a single *antecedent* and *consequent* relationship. However, for an extraction approach to be applicable in practice, it needs to comprehend also more complex relations containing at least two to three and at best an arbitrary number of *antecedents* and *consequents*. Understanding conjunctions, disjunctions, and negations is consequently imperative to fully capture the relationships between *antecedents* and *consequents*. We also found that *antecedents* and *consequents* need to be further decomposed into variable and condition to be suitable for our described use cases. Considering the combinatorics of *antecedents* and *consequents* and splitting them into variables and conditions is termed as fine-grained conditional extraction in our work.

Automated Conditional Extraction Facilitates Automated Acceptance Test Creation In Part III, we empirically proved that automated conditional extraction can indeed facilitate the realization of Use Case 1. We utilized `CiRA` to extract conditionals from requirements in fine-grained form and map them to a CEG, from which the minimal set of required test cases can be derived automatically. We demonstrated the feasibility of our approach on real-world data provided by *Allianz Deutschland AG* (insurance), *Ericsson* (telecommunication), and *Leopold Kostal GmbH & Co. KG* (automotive). Specifically, we compared our approach to the manual test case design. We showed that our approach is able to automatically generate 71.8 % of the 578 manually created test cases. In addition, our approach identified 80 relevant test cases that were missed in manual test case design. However, we also found that our approach does not substitute a test designer since domain knowledge is often necessary to identify all required test cases. Moreover, our study showed that a fully automated acceptance test generation is difficult to achieve as result of poor quality of the requirements. RE teams should therefore conduct a quality check of the requirements before processing them with our approach.

Conditionals Are a Source of Ambiguity When automatically extracting conditionals, it is crucial to acknowledge that conditionals are interpreted ambiguously by RE practitioners. We conducted a survey with 104 RE practitioners to understand how they logically interpret 12 different conditional clauses and found that practitioners disagree on whether an *antecedent* is only *sufficient* or also *necessary* for the *consequent*. There is a disagreement about the temporal occurrence of *antecedent* and *consequent* when different cue phrases are used. Driven by this observation, we integrated two variants of conditional extraction into our acceptance test creation approach (see Section 8.2). The first variant interprets conditionals as logical implications, while the second variant interprets them as equivalences and creates both positives and negative test cases. Users of our approach are then able to decide for themselves which variant corresponds to their logical interpretation. The explicit offering of both variants to the user fosters the discussion among project teams about the expected system behavior, by which misunderstandings can be resolved at an early stage.

Conclusion:

Authors of requirements often use conditionals to specify the desired system behavior. Therefore, conditionals contain rich semantic information about potential system inputs and expected system outputs. Automatically extracting conditionals bears a high potential for *Requirements Engineering* as it contributes to an increased automation of specific RE tasks. Our study with three industry partners proved that automated conditional extraction can indeed help to automatically extract acceptance tests. Further, besides assisting in automating RE tasks, automatic conditional extraction helps to identify and reduce misunderstandings in project teams. Since conditionals are interpreted differently by RE practitioners, teams must decide whether they consider *antecedents* to be only *sufficient* or also *necessary* for the *consequent*. We argue that automatically extracting conditionals from requirements and explicitly displaying corresponding positive and negative test cases to users can help to foster the discussion among practitioners.

11.1.2 How to Extract Conditionals From RE Artifacts?

Automated Conditional Extraction Is a Two-Step Problem Automated conditional extraction entails two distinct challenges: first, one needs to determine whether a requirement contains conditional statements. Only sentences containing conditionals are eligible for extraction, so sentences containing no conditional statements can be discarded. Second – if they contain conditionals – these conditionals need to be properly understood and extracted. In Part II, we introduced and compared different methods for the detection and extraction of conditionals from NL requirements.

Rule-Based Approaches Are Not Suitable for Automated Conditional Detection In Chapter 6, we compared the performance of rule-based, ML-based, and TL-based approaches in detecting conditionals in NL. We trained the approaches on 14,983 manually annotated sentences and found that rule-based systems that classify conditionals relying on the presence of certain cue phrases are not able distinguish between sentences that contain conditionals and sentences that do not. The best performance was achieved by syntactically enriched BERT embeddings combined with a softmax classifier (macro-F_1 score of 82 %). What needs to be acknowledged, however, is that the application of TL does not result in a large performance boost over conventional ML methods. In fact, our trained *Random Forest* classifier achieved a macro-F_1 score of 78 %. In summary, our evaluation showed that the detection of conditionals should not be limited to certain cue phrases. Rather, it requires consideration of the semantics of the whole sentence. BERT is well suited for this purpose and shows a slightly better performance than shallow ML models, which can be attributed to its ability to exploit the attention mechanism and its deep language understanding.

Automated Conditional Extraction Requires Deep Language Understanding In Chapter 7, we presented three approaches for the fine-grained extraction of conditionals from NL requirements. Our first approach is based on *Dependency Parsing* and uses recursive dependency matching to extract *antecedents* and *consequents* from a dependency tree.

Our second approach builds on the idea that conditionals represent recursive structures and uses a *Recursive Neural Tensor Network* to extract them in fine-grained form. Our third approach defines conditional extraction as a sequence labeling problem and uses different multi-class and multi-label classifiers to identify *antecedents* and *consequents*. We evaluated our approaches on different gold standard data sets (e.g., *Conditional Treebank* [6]) containing real-world requirements annotated in fine-grained form. Our evaluation demonstrates that the *Dependency Parsing*-based approach often fails to extract conditionals in case of grammatical errors in a NL sentence. Our RNTN-based approach shows a better performance in handling grammatical errors, while facing challenges in understanding the semantics of words that were not part of its training vocabulary. These problems render the approaches inapplicable in practice since (1) requirements regularly suffer from poor quality and (2) it is not possible to create a training corpus that includes a *"full"* vocabulary of all different domains. Our third approach addresses these issues and is more robust against grammatical errors and unknown words due to the usage of subword tokenization. In addition, it shows a significantly better performance in the detection of *antecedents* and *consequents* owing to the use of TL methods that have been pre-trained on large data sets. In fact, we found that a sigmoid classifier built on RoBERTa embeddings is best suited to extract conditionals in fine-grained form. It achieved a macro-F_1 score of 86 % when evaluated on a data set of 1,946 annotated conditionals.

`CiRA` - A Tool for the Automatic Extraction of Conditionals From RE Artifacts Based on the results of our experiments, we integrated the best model for conditional detection and the best model for conditional extraction into an end-to-end pipeline, called `CiRA`. We encourage fellow researchers and practitioners to use `CiRA` to extract conditionals from NL sentences in fine-grained form and utilize them for their individual use cases. So far, `CiRA` offers support for the automatic generation of acceptance tests based on the extracted conditionals and is publicly available at `http://www.cira.bth.se/demo/`. A detailed description of the functionality of `CiRA` can be found in Chapter 8.

Conclusion:

Automated extraction of conditionals is not a trivial task. Shallow rule-based systems are not suitable for extracting conditionals as they can be expressed in many different forms that are difficult to cover with patterns. Our studies proved that ML-based and TL-based approaches are better suited for determining conditionals in NL sentences and extracting them in fine-gained form. However, simply using ML and TL does not automatically lead to a solution of an NLP problem. Rather, the choice of an adequate ML and TL model is dependent on the context and the complexity of the problem that needs to be solved. This is particularly evident in our comparison of ML and TL models for the detection of conditionals. We did not observe a great deviation in performance between the best ML model and our best TL model in solving this binary classification problem. The benefits of *Transfer Learning* were most noticeable when dealing with the considerably more complex problem of conditional extraction. Owing to pre-training on large corpora, the TL models acquired a strong language understanding and are therefore capable of reliably extracting conditionals in fine-grained form.

Figure 11.1: Automated Extraction of Two-Sentence Conditionals.

11.2 Limitations & Outlook

This section discusses possible improvements of the presented work and illustrates directions for further research. To this end, we stimulate future work by means of motivating examples and suggest a way how our ideas could be implemented.

Automated Extraction of Two-Sentence Conditionals `CiRA` is limited to single-sentence conditionals and is not able to extract conditional statements that span multiple sentences (i.e., *"A and B occur. Thus, C evaluates to false"*). Our study presented in Chapter 4 revealed that two-sentence conditionals may arise in practice (e.g., indicated by cue phrases such as *"therefore"*, *"hence"*), requiring us to enhance `CiRA` in future work.

Motivating Example So far, `CiRA` is not capable of processing sentences like *"The user has admin rights. Therefore, he/she is allowed to access folders that include confidential information"*. Specifically, our approach can only detect *antecedents* and *consequents* contained in the same sentence. In the given example, the *antecedent* is contained in the first sentence and the *consequent* in the consecutive sentence. Consequently, we need to extract them as illustrated in Figure 11.1.

Implementation Proposal We hypothesize that our existing approach for the extraction of conditionals can be extended to two-sentence conditionals with seemingly little effort. In a first step, our created training corpus must be augmented with annotated two-sentence conditionals. The annotation scheme presented in Section 7.3.2 can be reused for this purpose, since we also want to annotate the two-sentence conditionals in fine-grained form. The annotation process may be more complex than the labeling of single-sentence conditionals, since the tag sequence is most likely longer and the *antecedents* and *consequents* are located in different sentences. It is crucial that we augment the training corpus with two-sentence conditionals that are expressed by different cue phrases. Otherwise, the extractor will operate with a bias towards certain cue phrases and will not be able to handle two-sentence conditionals in different forms. In a second step, we need to re-train our approach on the expanded training corpus. In this context, the annotated two-sentence conditionals need to be analyzed in order to adjust the maximum length of tokens per sentence for the TL models. So far, we set the maximum length to 80 tokens (see Section 7.3.3). This setting will most likely need to be updated with respect to the newly added training instances.

Isolated vs. Joint CEG Modeling In Chapter 10, we demonstrated how `CiRA` can be combined with *Cause-Effect-Graphing* to create acceptance tests automatically. So far, our approach only allows the creation of acceptance tests for a single requirement. Test

designers need to consider the relationships between requirements when creating a test suite in order to determine, for example, whether related requirements can be covered by joint test cases. The necessity of considering requirements as a coherent set, rather than in isolation, becomes particularly apparent when an existing test suite has to be adapted to evolving requirements. Let us assume new requirements have been implemented and need to be tested. In case requirements are considered separately, individual tests are created for each new requirement leading to a strong linear correlation between the number of requirements and tests [318]. This raises a number of problems: First, the number of test cases increases strictly with the addition of new requirements, which causes an increasing testing effort. Second, redundancies emerge within the test suite, as it is not checked whether existing test cases can already cover parts of new requirements or only need to be adapted. As a result, it cannot be guaranteed that the test suite is minimal. We argue that our presented approach contributes to the adaptation of test cases to changing requirements if we change the way the CEG is created. Our core idea is to examine the relationships between requirements during the test creation process and build *joint CEG models*. In the following, we illustrate our vision by an example and compare it with our current test generation process, which treats requirements separately.

Motivating Example Figure 11.2 describes three potential changes in a set of user stories: adding new acceptance criteria to existing user stories (see ACC 1.2), changing existing acceptance criteria in existing user stories (see ACC 2.2), and adding completely new user stories (see US 3). Initially, US 1 contains one acceptance criterion defining system behavior on the basis of two *antecedents* and *consequents*. Translating ACC 1.1 into a single CEG (see I.) and deriving test cases from it results in three test cases. As the project proceeds, the acceptance criterion ACC 1.2 is added to US 1. Considering both acceptance criteria in isolation and creating another individual CEG (see II.), two more test cases are added to the test suite. When analyzing both acceptance criteria, it is striking that they describe the same *consequents* c_1 and c_2 and are thus related to each other. Such dependencies can exist not only within a single user story, but also across multiple user stories as shown in US 2. ACC 2.2 is modified by inserting a new *antecedent* a_5 that incorporates the opposite of *consequent* c_1 which is also described in ACC 1.1 and ACC 1.2. A similar picture emerges for US 3 which has been newly created. It contains references between its own acceptance criteria and to acceptance criteria from other user stories. Creating individual *Cause-Effect-Graphs* (see I.-V.) for each acceptance criteria and deriving test cases from them yields $|T_1| = 13$. The test suite is therefore constantly growing due to the isolated consideration of acceptance criteria.

We argue that respecting dependencies between acceptance criteria when deriving test cases ensures to keep the test suite minimal while the requirements coverage remains the same. Therefore, rather than creating individual *Cause-Effect-Graphs* and converting them into test cases, we propose to create a joint CEG for related acceptance criteria. We hypothesize that this will allow to merge interrelated test cases and thus to optimize the test suite. The joint CEG for all related acceptance criteria of all user stories is shown in VI. Applying BPST to this CEG yields $|T_2| = 5$. This decrease of required test cases stems from the fact that several test cases from T_1 are merged because their *antecedents* and *consequents* overlap, e.g., t_1, t_4, t_7,

t_9, and t_{13} from T_1 are covered by t_1 in T_2. Furthermore, not all *consequents* have to be checked because they are also indirectly covered, e.g., c_1 and c_4 are covered by c_5. Nonetheless, if the test case t_1 from T_2 fails due to a mismatch of a final *consequent* like c_5, the explicit *antecedent* (e.g., a mismatch of c_1 from t_1 in T_1) is still traceable by adding the columns of the remaining *consequents* to T_2.

Consequently, T_1 and T_2 cover the same functionality, however, the CEG transformations lead to a significant minimization of the test cases without compromising the fault detection capability.

Implementation Proposal In order to realize our outlined vision, a number of challenges need to be overcome. We need to integrate antonym detection into the `CiRA` pipeline, which can be illustrated by the following example. Two acceptance criteria are related to each other if they refer to the same entities (see ACC 1.1 and ACC 3.2). Both deal with a certain state of the entity *"process model"*. While ACC 1.1 defines the state of the *"process model"* as a *consequent* which is set to *"faulty"* respectively *"incorrect"* when a_1 and a_2 occur, the negation of this state is used as an *antecedent* in ACC 3.2. For human reading comprehension, it is obvious that *"correct"* represents an antonym of *"faulty"*. Antonym detection, however, still represents a challenge in NLP [333]. Nevertheless, it is essential for the automated detection of related acceptance criteria and to ultimately build a joint CEG. In addition, we need to detect and interpret co-references. In linguistics, the use of co-references is very common to describe entities and their behavior. Co-references are expressions that refer to the same entity. An example is ACC 2.1 and ACC 2.2, where *"this option"* refers to the *"option"* introduced in ACC 2.1. Further examples are the use of pronouns such as *"it"* to reference entities of similar gender. Co-reference resolution is essential for the recognition of the dependencies between the acceptance criteria and must therefore be integrated into the `CiRA` pipeline. In recent years, antonym detection and co-reference resolution have attracted increasing research attention. To date, the state of the art method for co-reference resolution is the approach developed by Lee et al. [334], while Ono et al. [335] has developed the most performant method for antonym detection. Both methods achieved impressive performance gains compared to previous approaches and provide a good basis for a prototypical implementation of our instrument.

Furthermore, future research needs to study how related *Cause-Effect-Graphs* can be merged to realize our vision of joint *Cause-Effect-Graphs*. Specifically, we need a method that identifies interrelated *Cause-Effect-Graphs* from a set of arbitrary *Cause-Effect-Graphs* and transforms them using model-to-model-transformations. In doing so, it is crucial that the combinatorics of *antecedents* and *consequents* defined by the acceptance criteria are respected, as the test cases will otherwise check for incorrect system functionality. We hypothesize that *Cause-Effect-Graphs* that describe the same combination of entity and property (see I. and II. in Figure 11.2) or the same entities but with a negated property (see I. and III.) can be combined.

User Feedback Analysis by Using Automated Conditional Extraction Project teams must constantly adapt their software to the needs of their stakeholders in order to ensure user acceptance. In this context, the analysis of user feedback plays a major role in

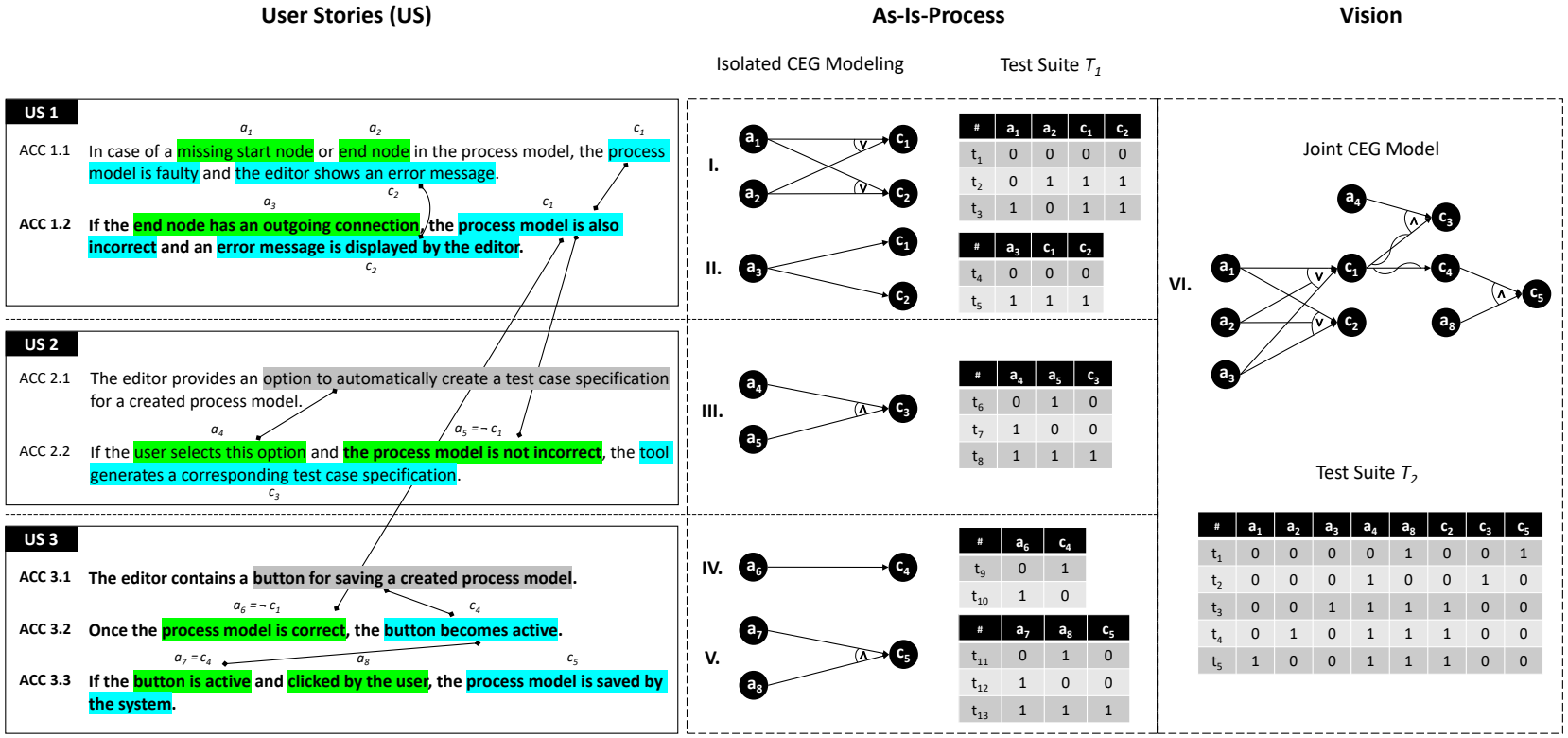


Figure 11.2: Generation of T From ACC Using CEG Modeling. Dependencies Between ACC Are Indicated by Arrows. *Antecedents* a_l Are Marked Green, *consequents* c_m Blue. Changes in US Are Indicated by **Bold** Letters. In the Center, the Outcome of the Test Generation Is Shown When Treating ACC Separately. The Right Part Illustrates Our Vision of Incorporating Dependencies Between ACC Into the Test Generation by Means of a Joint CEG. Within Each t_n, 1 Indicates That a_l Resp. c_m Is Present and Vice Versa.

enabling project teams to get an overview of features that are well received, features that are still missing, and bugs that need to be addressed. There are already a number of approaches that automatically analyze user feedback in tweets and app stores reviews. So far, the focus has only been on classifying user feedback (e.g., does the review contain a bug report or a fixed request). We argue that automatic conditional extraction can help to analyze online feedback in a more fine-grained way. Future studies should therefore use our approach to extract *antecedents* and *consequents* from user feedback.

Motivating Example In the case of bug reports, automated conditional extraction allows to understand under which conditions certain bugs occur: *"when I press button X, the screen of the smartphone freezes and I can no longer use the app."* Development teams can thus understand where to start when they want to fix the bug. Of course, using conditional extraction does not replace clean debugging to identify all possible conditions that are triggering the bug. However, the extraction of conditionals offers a first clue. In the case of feature requests, conditional extraction can be used to determine in which scenarios users expect new features or how existing features need to be adapted to hitherto neglected conditions.

Implementation Proposal Our existing approach to automated extraction of conditionals does not need to be adapted for this new use case. Rather, we propose to conduct a preliminary analysis of the potential of conditional extraction for user feedback analysis. For this purpose, future work should apply `CiRA` to the data and annotations of Maalej et al. [336] and Stanik et al. [337] containing about 6k English app reviews and about 25k English and Italian tweets categorized as *Problem Report*, *Feature Request*, or irrelevant. In a next step, we plan to present the extracted *antecedents* and *consequents* to developers and discuss about their expectations of a suitable conditional extraction for feedback analysis. In particular, we want to understand the added value that automated conditional extraction can provide to their use cases. Further, we want to determine which graphical representation is adequate to present the extracted conditionals to the practitioners in an easily understandable form.

Bibliography

[1] J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, and D. Freudenstein, "Specmate: Automated creation of test cases from acceptance criteria," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 321–331, 2020.

[2] J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, "Towards causality extraction from requirements," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 388–393, 2020.

[3] J. Fischbach, H. Femmer, D. Mendez, D. Fucci, and A. Vogelsang, "What makes agile test artifacts useful? an activity-based quality model from a practitioners' perspective," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[4] J. Fischbach, J. Frattini, A. Spaans, M. Kummeth, A. Vogelsang, D. Mendez, and M. Unterkalmsteiner, "Automatic detection of causality in requirement artifacts: The cira approach," in *Requirements Engineering: Foundation for Software Quality* (F. Dalpiaz and P. Spoletini, eds.), (Cham), pp. 19–36, Springer International Publishing, 2021.

[5] J. Fischbach, J. Frattini, D. Mendez, M. Unterkalmsteiner, H. Femmer, and A. Vogelsang, "How do practitioners interpret conditionals in requirements?," in *Product-Focused Software Process Improvement* (L. Ardito, A. Jedlitschka, M. Morisio, and M. Torchiano, eds.), (Cham), pp. 85–102, Springer International Publishing, 2021.

[6] J. Fischbach, T. Springer, J. Frattini, H. Femmer, A. Vogelsang, and D. Mendez, "Fine-grained causality extraction from natural language requirements using recursive neural tensor networks," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 60–69, 2021.

[7] J. Frattini, J. Fischbach, D. Mendez, M. Unterkalmsteiner, A. Vogelsang, and K. Wnuk, "Causality in requirements artifacts: prevalence, detection, and impact," *Requirements Engineering*, Feb 2022.

[8] C. Wiecher, J. Fischbach, J. Greenyer, A. Vogelsang, C. Wolff, and R. Dumitrescu, "Integrated and iterative requirements analysis and test specification: A case study at kostal," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 112–122, 2021.

[9] N. Jadallah, J. Fischbach, J. Frattini, and A. Vogelsang, "Cate: Causality tree extractor from natural language requirements," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 77–79, 2021.

[10] J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, and C. Wiecher, "Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study," 2022.

[11] J. Fischbach, M. Junker, A. Vogelsang, and D. Freudenstein, "Automated generation of test models from semi-structured requirements," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pp. 263–269, 2019.

[12] P. R. Henao, J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer learning for mining feature requests and bug reports from tweets and app store reviews," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 80–86, 2021.

[13] J. Bohn, J. Fischbach, M. Schmitt, H. Schütze, and A. Vogelsang, "Semi-automated labeling of requirement datasets for relation extraction," in *Proceedings of the 14th Workshop on Building and Using Comparable Corpora (BUCC 2021)*, (Online (Virtual Mode)), pp. 40–45, INCOMA Ltd., Sept. 2021.

[14] R. Bhatt and R. Pancheva, *Conditionals*, ch. 16, pp. 638–687. John Wiley & Sons, Ltd, 2006.

[15] P. Johnson-Laird and R. M. J. Byrne, "Conditionals: A theory of meaning, pragmatics, and inference," *Psychological Review*, vol. 109, no. 4, pp. 646–678, 2002.

[16] K. Fintel, "Exceptive conditionals: The meaning of unless," *North East Linguistics Society*, vol. 22, 1992.

[17] R. Declerck and S. Reed, *Conditionals : a comprehensive empirical analysis*. Berlin; New York: Mouton de Gruyter, 2001.

[18] R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik, *A Grammar of Contemporary English*. London Longman, 1972.

[19] F. Jackson, "On assertion and indicative conditionals," *The Philosophical Review*, vol. 88, no. 4, pp. 565–589, 1979.

[20] R. Girju, "Automatic detection of causal relations for question answering," in *Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering - Volume 12*, MultiSumQA '03, (USA), p. 76–83, Association for Computational Linguistics, 2003.

[21] C. Silverstein, S. Brin, R. Motwani, and J. Ullman, "Scalable techniques for mining causal structures," *Data Mining and Knowledge Discovery*, vol. 4, pp. 163–192, Jul 2000.

[22] M. Riaz and R. Girju, "Another look at causality: Discovering scenario-specific contingency relationships with no supervision," in *2010 IEEE Fourth International Conference on Semantic Computing*, pp. 361–368, 2010.

[23] C. Hashimoto, K. Torisawa, J. Kloetzer, M. Sano, I. Varga, J.-H. Oh, and Y. Kidawara, "Toward future scenario generation: Extracting event causality exploiting semantic relation, context, and association features," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Baltimore, Maryland), pp. 987–997, Association for Computational Linguistics, June 2014.

[24] J. Qiu, L. Xu, J. Zhai, and L. Luo, "Extracting causal relations from emergency cases based on conditional random fields," *Procedia Computer Science*, vol. 112, pp. 1623–1632, 2017. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France.

[25] Q.-C. Bui, B. Ó. Nualláin, C. A. Boucher, and P. M. Sloot, "Extracting causal relations on hiv drug resistance from literature," *BMC Bioinformatics*, vol. 11, p. 101, Feb 2010.

[26] C. S. G. Khoo, S. Chan, and Y. Niu, "Extracting causal knowledge from a medical database using graphical patterns," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, (USA), p. 336–343, Association for Computational Linguistics, 2000.

[27] C. Mihăilă and S. Ananiadou, "Semi-supervised learning of causal relations in biomedical scientific discourse," *BioMedical Engineering OnLine*, vol. 13, p. S1, Dec 2014.

[28] C. S. Khoo, S. H. Myaeng, and R. N. Oddy, "Using cause-effect relations in text to improve information retrieval precision," *Information Processing & Management*, vol. 37, no. 1, 2001.

[29] S. Doan, E. W. Yang, S. S. Tilak, P. W. Li, D. S. Zisook, and M. Torii, "Extracting health-related causality from twitter messages using natural language processing," *BMC Medical Informatics and Decision Making*, vol. 19, 2019.

[30] K. Radinsky, S. Davidovich, and S. Markovitch, "Learning causality for news events prediction," in *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, (New York, NY, USA), p. 909–918, Association for Computing Machinery, 2012.

[31] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, Incorporated, 1st ed., 2010.

[32] M. Glinz, "A glossary of requirements engineering terminology," tech. rep., International Requirements Engineering Board and University of Zurich, 2020.

[33] B. Boehm and P. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[34] W. Kuffel, "Extra time saves money," *Comput. Lang.*, 1990.

[35] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[36] R. Saini, G. Mussbacher, J. L. C. Guo, and J. Kienzle, "Automated traceability for domain modelling decisions empowered by artificial intelligence," in *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pp. 173–184, 2021.

[37] C. Stanik, T. Pietz, and W. Maalej, "Unsupervised topic discovery in user comments," in *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pp. 150–161, 2021.

[38] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "Norbert: Transfer learning for requirements classification," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 169–179, 2020.

[39] D. Torre, S. Abualhaija, M. Sabetzadeh, L. Briand, K. Baetens, P. Goes, and S. Forastier, "An ai-assisted approach for checking the completeness of privacy policies against gdpr," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 136–146, 2020.

[40] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: The best is yet to come," *IEEE Software*, vol. 35, no. 5, 2018.

[41] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software*, vol. 172, p. 110851, 2021.

[42] F. Gomes De Oliveira Neto, J. Horkoff, E. Knauss, R. Kasauli, and G. Liebel, "Challenges of aligning requirements engineering and system testing in large-scale agile: A multiple case study," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pp. 315–322, 2017.

[43] S. Wagner, D. M. Fernández, M. Felderer, A. Vetrò, M. Kalinowski, R. Wieringa, D. Pfahl, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayebi, M. Oivo, B. Penzenstadler, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spínola, A. Tuzcu, J. L. D. L. Vara, and D. Winkler, "Status quo in requirements engineering: A theory and a global family of surveys," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, feb 2019.

[44] D. M. Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M. T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. De La Vara, and R. Wieringa, "Naming the pain in requirements engineering," *Empirical Softw. Engg.*, vol. 22, no. 5, p. 2298–2338, 2017.

[45] M. Bano, M. Bano, D. Zowghi, A. Ferrari, P. Spoletini, and B. Donati, "Learning from mistakes: An empirical study of elicitation interviews performed by

novices," in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 182–193, 2018.

[46] M. D. Pickard and C. A. Roster, "Using computer automated systems to conduct personal interviews: Does the mere presence of a human face inhibit disclosure?," *Comput. Hum. Behav.*, vol. 105, apr 2020.

[47] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2016.

[48] E. Guzman, R. Alkadhi, and N. Seyff, "A needle in a haystack: What do twitter users say about software?," in *IEEE International Requirements Engineering Conference (RE)*, pp. 96–105, 2016.

[49] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, p. 291–300, 2015.

[50] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *IEEE International Requirements Engineering Conference (RE)*, pp. 125–134, 2013.

[51] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their ides," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), p. 179–190, Association for Computing Machinery, 2015.

[52] P. P. Kulkarni and Y. Joglekar, "Generating and analyzing test cases from software requirements using nlp and hadoop," in *International Journal of Current Engineering and Technology*, vol. 4, 2014.

[53] E. Bjarnason, P. Runeson, M. Borg, M. Unterkalmsteiner, E. Engström, B. Regnell, G. Sabaliauskaite, A. Loconsole, T. Gorschek, and R. Feldt, "Challenges and practices in aligning requirements with verification and validation: A case study of six companies," *Empirical Softw. Engg.*, vol. 19, p. 1809–1855, dec 2014.

[54] J. C. S. Coutinho, W. L. Andrade, and P. D. L. Machado, "Requirements engineering and software testing in agile methodologies: A systematic mapping," in *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES 2019, (New York, NY, USA), p. 322–331, Association for Computing Machinery, 2019.

[55] V. Garousi, S. Bauer, and M. Felderer, "Nlp-assisted software testing: A systematic mapping of the literature," *Information and Software Technology*, vol. 126, p. 106321, 2020.

[56] C. Wang, F. Pastore, A. Goknil, and L. Briand, "Automatic generation of acceptance test cases from use case specifications: an nlp-based approach," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[57] G. Carvalho, F. Barros, F. Lapschies, U. Schulze, and J. Peleska, "Model-based testing from controlled natural language requirements," in *Formal Techniques for Safety-Critical Systems*, pp. 19–35, 2014.

[58] F. A. Barros, L. Neves, E. Hori, and D. Torres, "The ucsCNL: A Controlled Natural Language for Use Case Specifications," in *SEKE*, pp. 250–253, 2011.

[59] S. Liu and S. Nakajima, "Automatic test case and test oracle generation based on functional scenarios in formal specifications for conformance testing," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[60] R. Sharma and K. K. Biswas, "Automated generation of test cases from logical specification of software requirements," in *ENASE*, pp. 241–248, 2014.

[61] L. Mich, M. Franch, and P. Novi Inverardi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, p. 40–56, 2004.

[62] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," *Innovations in Systems and Software Engineering*, vol. 10, p. 235–241, 2014.

[63] D. Freudenstein, M. Junker, J. Radduenz, S. Eder, and B. Hauptmann, "Automated test-design from requirements - the specmate tool," in *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*, pp. 5–8, 2018.

[64] K. Nursimulu and R. L. Probert, "Cause-effect graphing analysis and validation of requirements," in *Proceedings of the 1995 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '95, p. 46, IBM Press, 1995.

[65] A. Vogelsang, "Feature dependencies in automotive software systems: Extent, awareness, and refactoring," *Journal of Systems and Software*, vol. 160, 2020.

[66] S. Bohner, "Software change impacts-an evolving perspective," in *International Conference on Software Maintenance, 2002. Proceedings.*, pp. 263–272, 2002.

[67] Å. G. Dahlstedt and A. Persson, *Requirements Interdependencies: State of the Art and Future Challenges*, pp. 95–116. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[68] D.-S. Chang and K.-S. Choi, "Causal relation extraction using cue phrase and lexical pair probabilities," in *Natural Language Processing – IJCNLP 2004* (K.-Y. Su, J. Tsujii, J.-H. Lee, and O. Y. Kwong, eds.), (Berlin, Heidelberg), pp. 61–70, Springer, Springer Berlin Heidelberg, 2004.

[69] B. Rink and S. Harabagiu, "UTD: Classifying semantic relations by combining lexical and semantic resources," in *Proceedings of the 5th International Workshop on Semantic Evaluation*, (Uppsala, Sweden), pp. 256–259, Association for Computational Linguistics, July 2010.

[70] T. Dasgupta, R. Saha, L. Dey, and A. Naskar, "Automatic extraction of causal relations from text using linguistically informed deep neural networks," in *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, (Melbourne, Australia), pp. 306–316, Association for Computational Linguistics, July 2018.

[71] Z. Li, Q. Li, X. Zou, and J. Ren, "Causality extraction based on self-attentive BiLSTM-CRF with transferred embeddings," *CoRR*, vol. abs/1904.07629, 2019.

[72] C. S. G. Khoo, J. Kornfilt, R. N. Oddy, and S. H. Myaeng, "Automatic Extraction of Cause-Effect Information from Newspaper Text Without Knowledge-based Inferencing," *Literary and Linguistic Computing*, vol. 13, no. 4, 1998.

[73] K.-J. Stol and B. Fitzgerald, "The abc of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, sep 2018.

[74] N. Mostafazadeh, A. Grealish, N. Chambers, J. Allen, and L. Vanderwende, "CaTeRS: Causal and temporal relation scheme for semantic annotation of event structures," in *Proceedings of the Fourth Workshop on Events*, (San Diego, California), pp. 51–61, Association for Computational Linguistics, June 2016.

[75] P. Wolff, "Representing causation," *Journal of Experiment Psychology: General*, 2007.

[76] E. Blanco, N. Castell, and D. Moldovan, "Causal relation extraction," in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, (Marrakech, Morocco), European Language Resources Association (ELRA), May 2008.

[77] R. Girju, D. I. Moldovan, *et al.*, "Text mining for causal relations.," in *FLAIRS conference*, pp. 360–364, 2002.

[78] J. K. Hartshorne, "What is implicit causality?," *Language, Cognition and Neuroscience*, vol. 29, no. 7, pp. 804–824, 2014.

[79] C. Garvey and A. Caramazza, "Implicit causality in verbs," *Linguistic Inquiry*, vol. 5, no. 3, pp. 459–464, 1974.

[80] N. Altman and M. Krzywinski, "Association, correlation and causation," *Nature Methods*, vol. 12, pp. 899–900, 10 2015.

[81] P. W. Holland, "Statistics and causal inference," *Journal of the American Statistical Association*, vol. 81, no. 396, pp. 945–960, 1986.

[82] J. Check and R. K. Schutt, *Causation and Experimental Design*, ch. 5, pp. 116–144. SAGE Publications, Inc., 2012.

[83] J. Dul, "Necessary condition analysis (NCA): Logic and methodology of "necessary but not sufficient" causality," *Organizational Research Methods*, 2016.

[84] D. Lewis, *Counterfactuals*. Blackwell Publishers, 1973.

[85] R. Sassower, *Causality and Correlation*, pp. 1–4. American Cancer Society, 2017.

[86] H. A. Simon, "Spurious correlation: A causal interpretation," *Journal of the American Statistical Association*, vol. 49, no. 267, pp. 467–479, 1954.

[87] J. Puga, M. Krzywinski, and N. Altman, "Points of significance: Bayesian networks," *Nature Methods*, vol. 12, pp. 799–800, Aug. 2015. Copyright: Copyright 2015 Elsevier B.V., All rights reserved.

[88] J. E. Taplin and H. Staudenmayer, "Interpretation of abstract conditional sentences in deductive reasoning," *Journal of Verbal Learning and Verbal Behavior*, vol. 12, no. 5, pp. 530–542, 1973.

[89] S. L. Marcus and L. J. Rips, "Conditional reasoning," *Journal of Verbal Learning and Verbal Behavior*, vol. 18, no. 2, pp. 199–223, 1979.

[90] H. Staudenmayer, *Understanding conditional reasoning with meaningful propositions*, pp. 55–79. Psychology Press, 1975.

[91] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems*, 1994.

[92] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, (New York, NY, USA), p. 411–420, Association for Computing Machinery, 1999.

[93] T. I. of Electrical and E. Engineers, "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[94] G. Kotonya and I. Sommerville, "Requirements engineering with viewpoints," *Software Engineering Journal*, vol. 11, pp. 5–18(13), January 1996.

[95] R. Mall, *Fundamentals of Software Engineering.* PHI Learning, 2018.

[96] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, (New York, NY, USA), p. 35–46, Association for Computing Machinery, 2000.

[97] J. Dörr, S. Adam, M. Eisenbarth, and M. Ehresmann, "Implementing requirements engineering processes: Using cooperative self-assessment and improvement," *IEEE Software*, vol. 25, no. 3, pp. 71–77, 2008.

[98] I. J. S. . Software and systems engineering, "Systems and software engineering — Life cycle processes — Requirements engineering," standard, International Organization for Standardization, Geneva, CH, 2018.

[99] International Software Testing Qualifications Board, "Istqb glossary," accessed: 04-18-2022.

[100] G. Melnik and F. Maurer, "Multiple perspectives on executable acceptance test-driven development," in *Agile Processes in Software Engineering and Extreme Programming* (G. Concas, E. Damiani, M. Scotto, and G. Succi, eds.), (Berlin, Heidelberg), pp. 245–249, Springer Berlin Heidelberg, 2007.

[101] J. Weiß, A. Schill, I. Richter, and P. Mandl, "Literature review of empirical research studies within the domain of acceptance testing," in *42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016, Limassol, Cyprus, August 31 - Sept. 2, 2016*, pp. 181–188, IEEE Computer Society, 2016.

[102] S. Park and F. Maurer, "A literature review on story test driven development," in *Agile Processes in Software Engineering and Extreme Programming* (A. Sillitti, A. Martin, X. Wang, and E. Whitworth, eds.), (Berlin, Heidelberg), pp. 208–213, Springer Berlin Heidelberg, 2010.

[103] S. Hotomski, *Supporting requirements and acceptance tests alignment during software evolution.* PhD thesis, University of Zurich, 2019.

[104] I. J. S. . Software and systems engineering, "Systems and software engineering — Vocabulary," standard, International Organization for Standardization, Geneva, CH, 2017.

[105] H. M. Sneed, "Testing against natural language requirements," in *Seventh International Conference on Quality Software (QSIC 2007)*, pp. 380–387, 2007.

[106] Y. Goldberg and G. Hirst, *Neural Network Methods in Natural Language Processing.* Morgan & Claypool Publishers, 2017.

[107] M. A. Covington, "A fundamental algorithm for dependency parsing," in *In Proceedings of the 39th Annual ACM Southeast Conference*, pp. 95–102, 2001.

[108] J. D. Choi, J. Tetreault, and A. Stent, "It depends: Dependency parser comparison using a web-based evaluation tool," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Beijing, China), pp. 387–396, Association for Computational Linguistics, July 2015.

[109] L. Tesnière, *Eléments de syntaxe structurale.* Klincksieck, 1959.

[110] R. Basak, S. Naskar, D. P. Pakray, and A. Gelbukh, "Recognizing textual entailment by soft dependency tree matching," *Computacion y Sistemas*, vol. 19, p. 685–700, 12 2015.

[111] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman, "Universal Dependencies v1: A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, (Portorož, Slovenia), pp. 1659–1666, European Language Resources Association (ELRA), May 2016.

[112] D. Jurafsky and J. H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Upper Saddle River, N.J.: Pearson Prentice Hall, 2009.

[113] J. Nivre, "An efficient algorithm for projective dependency parsing," in *Proceedings of the Eighth International Conference on Parsing Technologies*, (Nancy, France), pp. 149–160, Apr. 2003.

[114] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[115] Y.-J. Chu and T.-H. Liu, "On the shortest arborescence of a directed graph," *Science Sinica*, vol. 14, pp. 1396–1400, 1965.

[116] J. Edmonds, "Optimum branchings," *Journal of Research of the National Bureau of Standards B*, vol. 71, no. 4, p. 233– 240, 1967.

[117] R. McDonald and J. Nivre, "Characterizing the errors of data-driven dependency parsing models," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, (Prague, Czech Republic), pp. 122–131, Association for Computational Linguistics, June 2007.

[118] R. McDonald and J. Nivre, "Analyzing and integrating dependency parsers," *Computational Linguistics*, vol. 37, pp. 197–230, Mar. 2011.

[119] I. Titov and J. Henderson, "Fast and robust multilingual dependency parsing with a generative latent variable model," in *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pp. 947–951, 2007.

[120] X. Carreras, "Experiments with a higher-order projective dependency parser," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, (Prague, Czech Republic), pp. 957–961, Association for Computational Linguistics, June 2007.

[121] G. Attardi, F. Dell'Orletta, M. Simi, A. Chanev, and M. Ciaramita, "Multilingual dependency parsing and domain adaptation using DeSR," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, (Prague, Czech Republic), pp. 1112–1118, Association for Computational Linguistics, June 2007.

[122] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi, "Maltparser: A language-independent system for data-driven dependency parsing," *Natural Language Engineering*, vol. 13, pp. 95–135, 2005.

[123] M. Honnibal and I. Montani, *spaCy NLP library*, We use the newest version of the en_core_web_sm model in Sep'2020.

[124] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič, "Non-projective dependency parsing using spanning tree algorithms," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, (Vancouver, British Columbia, Canada), pp. 523–530, Association for Computational Linguistics, Oct. 2005.

[125] A. Martins, N. Smith, and E. Xing, "Concise integer linear programming formulations for dependency parsing," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, (Suntec, Singapore), pp. 342–350, Association for Computational Linguistics, Aug. 2009.

[126] T. Dozat and C. D. Manning, "Deep biaffine attention for neural dependency parsing," 2017.

[127] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 1373–1378, Association for Computational Linguistics, Sept. 2015.

[128] R. Socher, C. C.-Y. Lin, A. Y. Ng, and C. D. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (Madison, WI, USA), p. 129–136, Omnipress, 2011.

[129] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, (Seattle, Washington, USA), pp. 1631–1642, Association for Computational Linguistics, Oct. 2013.

[130] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, pp. 347–352 vol.1, 1996.

[131] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)* (J. Burstein, C. Doran, and T. Solorio, eds.), vol. abs/1810.04805, pp. 4171–4186, Association for Computational Linguistics, 2019.

[132] W. Yang, Y. Xie, A. Lin, X. Li, L. Tan, K. Xiong, M. Li, and J. Lin, "End-to-end open-domain question answering with BERTserini," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, (Minneapolis, Minnesota), pp. 72–77, Association for Computational Linguistics, June 2019.

[133] K. Hakala and S. Pyysalo, "Biomedical named entity recognition with multilingual BERT," in *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks*, (Hong Kong, China), pp. 56–61, Association for Computational Linguistics, Nov. 2019.

[134] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, (Hong Kong, China), pp. 3730–3740, Association for Computational Linguistics, Nov. 2019.

[135] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[136] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019.

[137] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[138] J. Blitzer, R. McDonald, and F. Pereira, "Domain adaptation with structural correspondence learning," in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, (Sydney, Australia), pp. 120–128, Association for Computational Linguistics, July 2006.

[139] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), vol. 26, Curran Associates, Inc., 2013.

[140] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[141] C. E. Osgood, G. J. Suci, and P. H. Tannenbaum, *The Measurement of Meaning*. University of Illinois Press, 1957.

[142] J. R. Firth, "A synopsis of linguistic theory," *Oxford: Philological Society*, 1957. Reprinted in F. Palmer (ed.)(1968). Studies in Linguistic Analysis 1930-1955. Selected Papers of J.R. Firth., Harlow: Longman.

[143] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *CoRR*, vol. abs/1607.04606, 2016.

[144] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," tech. rep., OpenAI, 2019.

[145] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152, 2012.

[146] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 1715–1725, Association for Computational Linguistics, Aug. 2016.

[147] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[148] G. J. Myers, *The Art of Software Testing*. Wiley, 1979.

[149] W. R. Elmendorf, "Cause-effect graphs in functional testing," tech. rep., IBM Systems Development Division, 1973.

[150] P. Liggesmeyer, *Software-Qualität - Testen, Analysieren und Verifizieren von Software, 2. Auflage*. Spektrum Akademischer Verlag, 2009.

[151] R. M. Byrne and P. Johnson-Laird, "'if' and the problems of conditional reasoning," *Trends in Cognitive Sciences*, vol. 13, no. 7, pp. 282–287, 2009.

[152] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *2009 17th IEEE International Requirements Engineering Conference*, pp. 317–322, 2009.

[153] T. Tahvonen and E. Uusitalo, "Easy approach to requirements syntax in nuclear power plant safety design," in *2018 1st International Workshop on Easy Approach to Requirements Syntax (EARS)*, pp. 1–2, 2018.

[154] A. Mavin Mav and P. Wilkinson, "Ten years of ears," *IEEE Software*, vol. 36, no. 5, pp. 10–14, 2019.

[155] B. Hauptmann, *Reducing System Testing Effort by Focusing on Commonalities in Test Procedures*. PhD thesis, Technical University of Munich, 2016.

[156] B. Hauptmann, M. Junker, S. Eder, L. Heinemann, R. Vaas, and P. Braun, "Hunting for smells in natural language tests," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 1217–1220, 2013.

[157] J. L. Elshoff, "A numerical profile of commercial PL/i programs," *Software: Practice and Experience*, vol. 6, pp. 505–525, Oct. 1976.

[158] D. E. Knuth, "An empirical study of FORTRAN programs," *Software: Practice and Experience*, vol. 1, pp. 105–133, Apr. 1971.

[159] H. J. Saal and Z. Weiss, "An empirical study of APL programs," *Computer Languages*, vol. 2, pp. 47–59, Jan. 1977.

[160] Y. Ueda, A. Ihara, T. Ishio, T. Hirao, and K. Matsumoto, "How are if-conditional statements fixed through peer codereview?," *IEICE Transactions on Information and Systems*, vol. E101.D, no. 11, pp. 2720–2729, 2018.

[161] D. Spinellis, *Code Quality - The Open Source Perspective*. Boston: Adobe Press, 2006.

[162] K. Pan, S. Kim, and E. J. Whitehead, "Toward an understanding of bug fix patterns," *Empirical Software Engineering*, vol. 14, pp. 286–315, Aug. 2008.

[163] M. Martinez, L. Duchien, and M. Monperrus, "Automatically extracting instances of code change patterns with ast analysis," in *2013 IEEE International Conference on Software Maintenance*, pp. 388–391, 2013.

[164] S. H. Tan, J. Yi, Yulis, S. Mechtaev, and A. Roychoudhury, "Codeflaws: a programming competition benchmark for evaluating automated program repair tools," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 180–182, 2017.

[165] V. A. Thompson, "Interpretational factors in conditional reasoning," *Mem. Cognit.*, vol. 22, pp. 742–758, Nov. 1994.

[166] J. S. B. T. Evans, *The Psychology of Deductive Reasoning*. Psychology Press, 1982.

[167] J. S. B. T. Evans, "The social and communicative function of conditional statements," *Mind & Society*, vol. 4, pp. 97–113, June 2005.

[168] P. C. Wason, "Reasoning about a rule," *Quarterly Journal of Experimental Psychology*, vol. 20, no. 3, pp. 273–281, 1968.

[169] G. Gebauer and D. Laming, "Rational choices in wason's selection task," *Psychological Research*, vol. 60, pp. 284–293, Nov 1997.

[170] P. W. Cheng and K. J. Holyoak, "Pragmatic reasoning schemas," *Cognitive Psychology*, vol. 17, pp. 391–416, Oct. 1985.

[171] V. Goel and R. J. Dolan, "Explaining modulation of reasoning by belief," *Cognition*, vol. 87, pp. B11–B22, Feb. 2003.

[172] E. Ohm and V. A. Thompson, "Everyday reasoning with inducements and advice," *Thinking & Reasoning*, vol. 10, pp. 241–272, Aug. 2004.

[173] N. L. Digdon, *Conditional And Biconditional Interpretations Of If-then Sentences: The Role Of Content And Context*. PhD thesis, University of Western Ontario, 1986.

[174] P. Pollard, "Human reasoning: Some possible effects of availability," *Cognition*, vol. 12, no. 1, pp. 65–96, 1982.

[175] W. Bucci, "The interpretation of universal affirmative propositions," *Cognition*, vol. 6, no. 1, pp. 55–77, 1978.

[176] H. Markovits, "Familiarity effects in conditional reasoning.," *Journal of Educational Psychology*, vol. 78, no. 6, pp. 492–494, 1986.

[177] H. Markovits, "Awareness of the 'possible' as a mediator of formal thinking in conditional reasoning problems," *British Journal of Psychology*, vol. 75, pp. 367–376, Aug. 1984.

[178] R. M. Byrne, "Suppressing valid inferences with conditionals," *Cognition*, vol. 31, no. 1, pp. 61–83, 1989.

[179] B. Rumain, J. Connell, and M. D. Braine, "Conversational comprehension processes are responsible for reasoning fallacies in children as well as adults: If is not the biconditional.," *Developmental Psychology*, vol. 19, no. 4, pp. 471–481, 1983.

[180] M. Oaksford, F. Morris, B. Grainger, and J. M. G. Williams, "Mood, reasoning, and central executive processes.," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 22, no. 2, pp. 476–492, 1996.

[181] R. J. Melton, "The role of positive affect in syllogism performance," *Personality and Social Psychology Bulletin*, vol. 21, pp. 788–794, Aug. 1995.

[182] I. Blanchette and A. Richards, "Reasoning about emotional and neutral materials: Is logic affected by emotion?," *Psychological Science*, vol. 15, pp. 745–752, Nov. 2004.

[183] A. Lefford, "The influence of emotonal subject matter on logical reasoning," *The Journal of General Psychology*, vol. 34, pp. 127–151, Apr. 1946.

[184] I. Blanchette, "The effect of emotion on interpretation and logic in a conditional reasoning task," *Memory & Cognition*, vol. 34, pp. 1112–1125, July 2006.

[185] J.-F. Bonnefon, V. Girotto, and P. Legrenzi, "The psychology of reasoning about preferences and unconsequential decisions," *Synthese*, vol. 185, pp. 27–41, May 2011.

[186] J. S. B. T. Evans, S. E. Newstead, and R. M. J. Byrne, *Human Reasoning - The Psychology of Deduction*. London: Psychology Press, 1993.

[187] J. S. Evans, H. Neilens, S. J. Handley, and D. E. Over, "When can we say 'if'?," *Cognition*, vol. 108, pp. 100–116, July 2008.

[188] E. Ohm and V. A. Thompson, "Conditional probability and pragmatic conditionals: Dissociating truth and effectiveness," *Thinking & Reasoning*, vol. 12, pp. 257–280, Aug. 2006.

[189] D. Edgington, "Do conditionals have truth-conditions?," *Cr'itica*, vol. 18, no. 52, pp. 3–30, 1986.

[190] J. Yang, S. C. Han, and J. Poon, "A survey on extraction of causal relations from natural language text," *CoRR*, vol. abs/2101.06426, 2021.

[191] N. Asghar, "Automatic extraction of causal relations from natural language texts: A comprehensive survey," *CoRR*, vol. abs/1605.07895, 2016.

[192] R. Grishman, *Domain modeling for language analysis*. Proceedings of the Conference at the University of Duisburg, March 1988, Peter Lang Verlag, 1988.

[193] D. Garcia, "Coatis, an nlp system to locate expressions of actions connected by causality links," in *Knowledge Acquisition, Modeling and Management* (E. Plaza and R. Benjamins, eds.), (Berlin, Heidelberg), pp. 347–352, Springer Berlin Heidelberg, 1997.

[194] C.-H. Wu, L.-C. Yu, and F.-L. Jang, "Using semantic dependencies to mine depressive symptoms from consultation records," *IEEE Intelligent Systems*, 2005.

[195] K. Chan and W. Lam, "Extracting causation knowledge from natural language texts," *International Journal of Intelligent Systems*, vol. 20, no. 3, pp. 327–358, 2005.

[196] T. Inui, K. Inui, and Y. Matsumoto, "Acquiring causal knowledge from text using the connective marker tame," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 4, no. 4, pp. 435–474, 2005.

[197] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, (Baltimore, Maryland), pp. 55–60, Association for Computational Linguistics, June 2014.

[198] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[199] Y. Sun, K. Xie, N. Liu, S. Yan, B. Zhang, and Z. Chen, "Causal relation of queries from temporal logs," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), p. 1141–1142, Association for Computing Machinery, 2007.

[200] S. Bethard and J. H. Martin, "Learning semantic links from a corpus of parallel temporal and causal relations," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, HLT-Short '08, (USA), p. 177–180, Association for Computational Linguistics, 2008.

[201] Z. Lin, M.-Y. Kan, and H. T. Ng, "Recognizing implicit discourse relations in the penn discourse treebank," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing Volume 1 - EMNLP '09*, Association for Computational Linguistics, 2009.

[202] A. Sorgente, G. Vettigli, and F. Mele, "Automatic extraction of cause-effect relations in natural language text," in *DART@AI*IA*, 2013.

[203] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[204] K. Mrini, F. Dernoncourt, Q. H. Tran, T. Bui, W. Chang, and N. Nakashole, "Rethinking self-attention: Towards interpretability in neural parsing," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, (Online), pp. 731–742, Association for Computational Linguistics, Nov. 2020.

[205] Y. Kirstain, O. Ram, and O. Levy, "Coreference resolution without span representations," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, (Online), pp. 14–19, Association for Computational Linguistics, Aug. 2021.

[206] S. Vashishth, P. Jain, and P. Talukdar, "Cesi: Canonicalizing open knowledge bases using embeddings and side information," in *Proceedings of the 2018 World Wide*

Web Conference, WWW '18, (Republic and Canton of Geneva, CHE), p. 1317–1327, International World Wide Web Conferences Steering Committee, 2018.

[207] X. Wang, Y. Jiang, N. Bach, T. Wang, Z. Huang, F. Huang, and K. Tu, "Automated concatenation of embeddings for structured prediction," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, (Online), pp. 2643–2660, Association for Computational Linguistics, Aug. 2021.

[208] Y. Xu, L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin, "Classifying relations via long short term memory networks along shortest dependency paths," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 1785–1794, Association for Computational Linguistics, Sept. 2015.

[209] E. M. Ponti and A. Korhonen, "Event-related features in feedforward neural networks contribute to identifying causal relations in discourse," in *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, (Valencia, Spain), pp. 25–30, Association for Computational Linguistics, Apr. 2017.

[210] R. Prasad, N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber, "The Penn Discourse TreeBank 2.0.," in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, (Marrakech, Morocco), European Language Resources Association (ELRA), May 2008.

[211] P. Aleixo, T. Alexandre, and S. Pardo, "Cstnews: um córpus de textos jornalísticos anotados segundo a teoria discursiva multidocumento cst (cross-document structure theory)," tech. rep., Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 05 2008.

[212] C. Kruengkrai, K. Torisawa, C. Hashimoto, J. Kloetzer, J.-H. Oh, and M. Tanaka, "Improving event causality recognition with multiple background knowledge sources using multi-column convolutional neural networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[213] Y. Zhang, P. Qi, and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 2205–2215, Association for Computational Linguistics, Oct.-Nov. 2018.

[214] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf, "FLAIR: An easy-to-use framework for state-of-the-art NLP," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, (Minneapolis, Minnesota), pp. 54–59, Association for Computational Linguistics, June 2019.

[215] M. Kyriakakis, I. Androutsopoulos, J. G. i Ametllé, and A. Saudabayev, "Transfer learning for causal sentence detection," *arXiv*, vol. abs/1906.07544, 2019.

[216] I. Hendrickx, S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, L. Romano, and S. Szpakowicz, "SemEval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals," in *SemEval*, pp. 33–38, 2010.

[217] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, (New York, NY, USA), p. 377–384, Association for Computing Machinery, 2006.

[218] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 502–505, 2017.

[219] I. Ahsan, W. H. Butt, M. A. Ahmed, and M. W. Anwar, "A comprehensive investigation of natural language processing techniques and tools to generate automated test cases," in *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[220] R. Koci, "Requirements modelling and software systems implementation using formal languages." International Conference on Software Engineering Advances (ICSEA), 2018.

[221] J. Woodcock and J. Davies, *Using Z - Specification, Refinement, and Proof*. London: Prentice Hall, 1996.

[222] J. R. Abrial, A. Hoare, and P. Chapron, "What is b?," in *The B-Book*, pp. xv–xxiv, Cambridge: Cambridge University Press, Oct. 1996.

[223] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, p. 223–252, sep 1977.

[224] D. Bjørner and C. B. Jones, eds., *The Vienna Development Method: The Meta-Language*, vol. 61 of *Lecture Notes in Computer Science*, Springer, 1978.

[225] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, pp. 297–312, Apr. 2011.

[226] R. Sharma and K. K. Biswas, "Automated generation of test cases from logical specification of software requirements," in *2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pp. 1–8, 2014.

[227] S. Liu, *Formal engineering for industrial software development*. Berlin, Germany: Springer, 2004 ed., Mar. 2013.

[228] S. Liu, *Formal Engineering for Industrial Software Development - Using the SOFL Method*. Wiesbaden: Springer Berlin Heidelberg, 2010.

[229] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[230] T. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 3, pp. 178–187, 1978.

[231] G. Tretmans and H. Brinksma, "Torx: Automated model-based testing," in *First European Conference on Model-Driven Software Engineering* (A. Hartman and K. Dussa-Ziegler, eds.), pp. 31–43, Dec. 2003. null ; Conference date: 11-12-2003 Through 12-12-2003.

[232] I. J. S. . Software and systems engineering, "Information processing systems — Open Systems Interconnection — LOTOS — A formal description technique based on the temporal ordering of observational behaviour," standard, International Organization for Standardization, Geneva, CH, 1989.

[233] G. J. Holzmann, *Design And Validation Of Computer Protocols*. Philadelphia, PA: Prentice Hall, Oct. 1990.

[234] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on Software Engineering*, vol. 17, pp. 591–603, June 1991.

[235] K. Sabnani and A. Dahbura, "A protocol test generation procedure," *Comput. Netw. ISDN Syst.*, vol. 15, p. 285–297, sep 1988.

[236] S. Helke, T. Neustupny, and T. Santen, "Automating test case generation from z specifications with isabelle," in *ZUM '97: The Z Formal Specification Notation* (J. P. Bowen, M. G. Hinchey, and D. Till, eds.), (Berlin, Heidelberg), pp. 52–71, Springer Berlin Heidelberg, 1997.

[237] S. Burton, "Automated testing from z specifications," tech. rep., Department of Computer Science, University of York, 2000.

[238] M. Satpathy, M. Butler, M. Leuschel, and S. Ramesh, "Automatic testing from formal specifications," in *Tests and Proofs* (Y. Gurevich and B. Meyer, eds.), (Berlin, Heidelberg), pp. 95–113, Springer Berlin Heidelberg, 2007.

[239] T. Kuhn, "A survey and classification of controlled natural languages," *Computational Linguistics*, vol. 40, pp. 121–170, Mar. 2014.

[240] G. Carvalho, D. Falcão, F. Barros, A. Sampaio, A. Mota, L. Motta, and M. Blackburn, "Nat2testscr: Test case generation from natural language requirements based on scr specifications," *Science of Computer Programming*, vol. 95, pp. 275–297, 2014. Special Section: ACM SAC-SVT 2013 + Bytecode 2013.

[241] M. R. Blackburn, R. D. Busser, and J. S. Fontaine, "Automatic generation of test vectors for scr-style specifications," in *Proc. 12th h Annual Conference on Computer Assurance*, pp. 54–67, 1997.

[242] A. L. L. de Figueiredo, W. L. Andrade, and P. D. L. Machado, "Generating interaction test cases for mobile phone systems from use case specifications," *SIGSOFT Softw. Eng. Notes*, vol. 31, p. 1–10, nov 2006.

[243] C. A. R. Hoare, "Communicating sequential processes," *Commun. ACM*, vol. 21, p. 666–677, aug 1978.

[244] S. Nogueira, A. Sampaio, and A. Mota, "Test generation from state based use case models," *Formal Aspects of Computing*, vol. 26, pp. 441–490, May 2014.

[245] P. Borba, D. Torres, R. Marques, and L. Wetzel, "Target - test and requirements generation tool," in *Motorola's 2007 Innovation Conference (IC'2007)*, 2007.

[246] M. Badri, L. Badri, and M. Naha, "A use case driven testing process: Towards a formal approach based on UML collaboration diagrams," in *Formal Approaches to Software Testing*, pp. 223–235, Springer Berlin Heidelberg, 2004.

[247] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jezequel, "Requirements by contracts allow automated system testing," in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pp. 85–96, 2003.

[248] I. L. Araújo, I. S. Santos, J. a. B. F. Filho, R. M. C. Andrade, and P. S. Neto, "Generating test cases and procedures from use cases in dynamic software product lines," in *Proceedings of the Symposium on Applied Computing*, SAC '17, (New York, NY, USA), p. 1296–1301, Association for Computing Machinery, 2017.

[249] C. Wang, F. Pastore, A. Goknil, L. C. Briand, and Z. Iqbal, "Umtg: A toolset to automatically generate system test cases from use case specifications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), p. 942–945, Association for Computing Machinery, 2015.

[250] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, (New York, NY, USA), p. 385–396, Association for Computing Machinery, 2015.

[251] M. Zhang, T. Yue, S. Ali, H. Zhang, and J. Wu, "A systematic approach to automatically derive test cases from use cases specified in restricted natural languages," in *SAM*, pp. 142–157, Springer International Publishing, 2014.

[252] T. Yue, S. Ali, and M. Zhang, "Rtcm: A natural language based, automated, and practical test case generation framework," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, (New York, NY, USA), p. 397–408, Association for Computing Machinery, 2015.

[253] E. Sarmiento, J. C. Sampaio do Prado Leite, and E. Almentero, "C&l: Generating model based test cases from natural language requirements descriptions," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, pp. 32–38, 2014.

[254] E. Sarmiento, J. C. Leite, E. Almentero, and G. Sotomayor Alzamora, "Test scenario generation from natural language requirements descriptions based on petri-nets," *Electronic Notes in Theoretical Computer Science*, vol. 329, pp. 123–148, 2016.

[255] D. North, "Introducing behaviour-driven development," *Better Software*, pp. 4–12, 2006.

[256] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[257] S. Rose, M. Wynne, and A. Hellesøy, *The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers*. O'Reilly UK, 2015.

[258] P. Rane, "Automatic generation of test cases for agile using natural language processing," Master's thesis, Virginia Tech University, 2017.

[259] A. K. Jena, S. K. Swain, and D. P. Mohapatra, "A novel approach for test case generation from uml activity diagram," in *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pp. 621–629, 2014.

[260] S. Masuda, T. Matsuodani, and K. Tsuda, "Syntactic rules of extracting test cases from software requirements," in *Proceedings of the 2016 8th International Conference on Information Management and Engineering*, ICIME 2016, (New York, NY, USA), p. 12–17, Association for Computing Machinery, 2016.

[261] A. Dwarakanath and S. Sengupta, "Litmus: Generation of test cases from functional requirements in natural language," in *Natural Language Processing and Information Systems* (G. Bouma, A. Ittoo, E. Métais, and H. Wortmann, eds.), (Berlin, Heidelberg), pp. 58–69, Springer Berlin Heidelberg, 2012.

[262] A. Goffi, A. Gorla, M. D. Ernst, and M. Pezzè, "Automatic generation of oracles for exceptional behaviors," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ACM, July 2016.

[263] V. A. d. Santiago Júnior and V. L. Vijaykumar, "Generating model-based test cases from natural language requirements for space application software," *Software Quality Journal*, vol. 20, no. 1, p. 77–143, 2012.

[264] R. P. Verma and M. R. Beg, "Generation of test cases from software requirements using natural language processing," in *2013 6th International Conference on Emerging Trends in Engineering and Technology*, pp. 140–147, 2013.

[265] A. Ansari, M. B. Shagufta, A. Sadaf Fatima, and S. Tehreem, "Constructing test cases using natural language processing," in *2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)*, pp. 95–99, 2017.

[266] R. Boddu, L. Guo, S. Mukhopadhyay, and B. Cukic, "Retna: from requirements to testing in a natural way," in *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, pp. 262–271, 2004.

[267] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The Penn Treebank: Annotating predicate argument structure," in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[268] P. Blackburn, J. Bos, M. Kohlhase, and H. De Nivelle, "Inference and computational semantics," in *Computing Meaning Volume II* (H. Bunt, R. Muskens, and E. Thijsse, eds.), pp. 11–28, Kluwer Academic Publishers, 2001.

[269] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, April 2009.

[270] C. Robson, *Real World Research - A Resource for Social Scientists and Practitioner-Researchers*. Blackwell Publishers, 2002.

[271] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning - a Guide to Corpus-Building for Applications*. O'Reilly Media, Inc., 2012.

[272] M. Neves and J. Seva, "An extensive review of tools for manual annotation of documents," *Briefings in Bioinformatics*, 2019.

[273] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, 1960.

[274] A. Viera and J. Garrett, "Understanding interobserver agreement: The kappa statistic," *Family medicine*, 2005.

[275] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia Medica*, 2012.

[276] A. R. Feinstein and D. V. Cicchetti, "High agreement but low kappa: I. the problems of two paradoxes," *Journal of Clinical Epidemiology*, 1990.

[277] N. Wongpakaran, T. Wongpakaran, D. Wedding, and K. Gwet, "A comparison of cohen's kappa and gwet's ac1 when calculating inter-rater reliability coefficients: A study conducted with personality disorder samples," *BMC Medical Research Methodology*, 2013.

[278] K. Gwet, *Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters*. Advanced Analytics, LLC, 2012.

[279] K. Gwet, *AgreeStat Analytics*, Cloud-based version (AgreeStat360) was used in Sep'2020.

[280] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, 1977.

[281] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.

[282] M. L. McHugh, "The chi-square test of independence," *Biochemia medica*, vol. 23, no. 2, pp. 143–149, 2013.

[283] S. C. Latta, C. A. Howell, M. D. Dettling, and R. L. Cormier, "Use of data on avian demographics and site persistence during overwintering to assess quality of restored riparian habitat," *Conservation Biology*, vol. 26, no. 3, pp. 482–492, 2012.

[284] J. Frattini, M. Junker, M. Unterkalmsteiner, and D. Mendez, "Automatic extraction of cause-effect-relations from requirements artifacts," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 561–572, 2020.

[285] M. Ciolkowski, O. Laitenberger, S. Vegas, and S. Biffl, *Practical Experiences in the Design and Conduct of Surveys in Empirical Software Engineering*, pp. 104–128. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[286] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," 2020.

[287] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[288] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Nlp tool showcase at nlp4re," in *Requirements Engineering: Foundation for Software Quality*, 2019.

[289] D. A. Dillman, J. D. Smyth, and L. M. Christian, *Internet, Phone, Mail, and Mixed-Mode Surveys: The Tailored Design Method*. John Wiley & Sons, Ltd, 2014.

[290] QuestBack AG, "Unipark / enterprise feedback suite." `https://www.unipark.com/`, 2021-02-25.

[291] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Academic Press, 1988.

[292] L. C. Freeman, *Elementary Applied Statistics: For Students in Behavioral Science*. John Wiley & Sons, Ltd, 1965.

[293] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, 2000.

[294] J. L. Fleiss, B. Levin, and M. C. Paik, *The Measurement of Interrater Agreement*, ch. 18, pp. 598–626. John Wiley & Sons, Ltd, 2003.

[295] D. Yates, D. Moore, and G. McCabe, *The Practice of Statistics*. W. H. Freeman, 1999.

[296] D. M. Berry and M. M. Krieger, "From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0," 2000.

[297] B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, "Using nlp to detect requirements defects: An industrial experience in the railway domain," in *Requirements Engineering: Foundation for Software Quality* (P. Grünbacher and A. Perini, eds.), (Cham), pp. 344–360, Springer International Publishing, 2017.

[298] D. Sundararaman, V. Subramanian, G. Wang, S. Si, D. Shen, D. Wang, and L. Carin, "Syntax-infused transformer and bert models for machine translation and natural language understanding," 2019.

[299] M. Fares, A. Kutuzov, S. Oepen, and E. Velldal, "Word vectors, reuse, and replicability: Towards a community repository of large-text resources," in *Proceedings of the 21st Nordic Conference on Computational Linguistics*, (Gothenburg, Sweden), pp. 271–276, Association for Computational Linguistics, May 2017.

[300] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014.

[301] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "brat: a web-based tool for NLP-assisted text annotation," in *EACL*, 2021.

[302] T. Kolditz, C. Lohr, J. Hellrich, L. Modersohn, B. Betz, M. Kiehntopf, and U. Hahn, "Annotating german clinical documents for de-identification," *Studies in health technology and informatics*, pp. 203–207, 2019.

[303] G. Hripcsak and A. S. Rothschild, "Agreement, the f-measure, and reliability in information retrieval," *J. Am. Med. Inform. Assoc.*, p. 296–298, 2005.

[304] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* O'Reilly Media, Inc., 2009.

[305] J. Xu, W. Zuo, S. Liang, and X. Zuo, "A review of dataset and labeling methods for causality extraction," in *COLING*, pp. 1519–1531, 2020.

[306] R. Girju, P. Nakov, V. Nastase, S. Szpakowicz, P. Turney, and D. Yuret, "Semeval-2007 task 04: Classification of semantic relations between nominals," in *SemEval*, p. 13–18, 2007.

[307] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[308] H. Femmer and A. Vogelsang, "Requirements quality is quality in use," *IEEE Software*, vol. 36, no. 3, pp. 83–91, 2019.

[309] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, J. Fowler, M.and Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001.

[310] K. Schwaber, "Scrum development process," in *Business Object Design and Implementation* (J. Sutherland, C. Casanave, J. Miller, P. Patel, and G. Hollowell, eds.), (London), pp. 117–134, Springer London, 1997.

[311] C. J. Stettina and W. Heijstek, "Necessary and neglected? an empirical study of internal documentation in agile software development teams," in *Proceedings of the 29th ACM International Conference on Design of Communication*, SIGDOC '11, (New York, NY, USA), p. 159–166, Association for Computing Machinery, 2011.

[312] J. Bass, "Artefacts and agile method tailoring in large-scale offshore software development programmes," *Information and Software Technology*, vol. 75, 2016.

[313] O. Liskin, "How artifacts support and impede requirements communication," in *Requirements Engineering: Foundation for Software Quality* (S. A. Fricker and K. Schneider, eds.), (Cham), pp. 132–147, Springer International Publishing, 2015.

[314] G. Wagenaar, R. Helms, D. Damian, and S. Brinkkemper, "Artefacts in agile software development," in *Product-Focused Software Process Improvement* (P. Abrahamsson, L. Corral, M. Oivo, and B. Russo, eds.), (Cham), pp. 133–148, Springer International Publishing, 2015.

[315] G. Wagenaar, S. Overbeek, G. Lucassen, S. Brinkkemper, and K. Schneider, "Working software over comprehensive documentation – rationales of agile teams for artefacts usage," *Journal of Software Engineering Research and Development*, vol. 6, 2018.

[316] International Software Testing Qualifications Board, "Certified tester specialist syllabus," 2019.

[317] I. J. S. . Software and systems engineering, "Software and systems engineering — software testing — part 1: General concepts," standard, International Organization for Standardization, 2022.

[318] S. Hotomski, E. B. Charrada, and M. Glinz, "An exploratory study on handling requirements and acceptance test documentation in industry," in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pp. 116–125, 2016.

[319] V. R. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Approach," *Encyclopedia of Software Engineering*, vol. 1, 1994.

[320] M. Q. Patton, *Qualitative evaluation and research methods, 2nd ed.* Qualitative evaluation and research methods, 2nd ed., Thousand Oaks, CA, US: Sage Publications, Inc, 1990.

[321] P. Mayring, *Qualitative Content Analysis: Theoretical Background and Procedures*, pp. 365–380. Dordrecht: Springer Netherlands, 2015.

[322] D. M. Berry and E. Kamsties, *Ambiguity in Requirements Specification*, pp. 7–44. Boston, MA: Springer US, 2004.

[323] V. Gervasi, A. Ferrari, D. Zowghi, and P. Spoletini, *Ambiguity in Requirements Engineering: Towards a Unifying Framework*, pp. 191–210. Cham: Springer International Publishing, 2019.

[324] D. M. Fernández and S. Wagner, "Naming the pain in requirements engineering: A design for a global family of surveys and first results from germany," *Information and Software Technology*, vol. 57, 2015.

[325] Y. Wand and R. Y. Wang, "Anchoring data quality dimensions in ontological foundations," *Communications of the ACM*, vol. 39, 1996.

[326] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *J. Manage. Inf. Syst.*, vol. 12, 1996.

[327] M. Bovee, R. P. Srivastava, and B. Mak, "A conceptual framework and belief-function approach to assessing overall information quality," *International Journal of Intelligent Systems*, vol. 18, 2001.

[328] T. Catarci and M. Scannapieco, "Data quality under the computer science perspective," *Archivi & Computer*, vol. 2, 2003.

[329] B. Kitchenham and S. L. Pfleeger, "Principles of survey research: Part 5: Populations and samples," *SIGSOFT Softw. Eng. Notes*, vol. 27, no. 5, p. 17–20, 2002.

[330] A. Kumar, P. Makhija, and A. Gupta, "Noisy text data: Achilles' heel of BERT," in *W-NUT*, pp. 16–21, 2020.

[331] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[332] H. K. V. Tran, M. Unterkalmsteiner, J. Börstler, and N. b. Ali, "Assessing test artifact quality—a tertiary study," *Inf. Softw. Technol.*, vol. 139, 2021.

[333] S. Mohammad, B. J. Dorr, G. Hirst, and P. D. Turney, "Computing lexical contrast," *CoRR*, vol. abs/1308.6300, 2013.

[334] K. Lee, L. He, M. Lewis, and L. Zettlemoyer, "End-to-end neural coreference resolution," 2017.

[335] M. Ono, M. Miwa, and Y. Sasaki, "Word embedding-based antonym detection using thesauri and distributional information," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 984–989, 2015.

[336] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requir. Eng.*, vol. 21, no. 3, p. 311–331, 2016.

[337] C. Stanik, M. Haering, and W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in *IEEE International Requirements Engineering Conference Workshops (REW)*, pp. 220–226, 2019.

Erklärung zur Dissertation

"Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel und Literatur angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind als solche kenntlich gemacht. Ich versichere an Eides statt, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen und eingebundenen Artikeln und Manuskripten - noch nicht veröffentlicht worden ist sowie, dass ich eine Veröffentlichung der Dissertation vor Abschluss der Promotion nicht ohne Genehmigung des Promotionsausschusses vornehmen werde. Die Bestimmungen dieser Ordnung sind mir bekannt. Darüber hinaus erkläre ich hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten der Universität zu Köln gelesen und sie bei der Durchführung der Dissertation zugrundeliegenden Arbeiten und der schriftlich verfassten Dissertation beachtet habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht."

Teilpublikationen:

[1]: J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, and D. Freudenstein, "Specmate: Automated creation of test cases from acceptance criteria," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 321–331, 2020

[2]: J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, "Towards causality extraction from requirements," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 388–393, 2020

[3]: J. Fischbach, H. Femmer, D. Mendez, D. Fucci, and A. Vogelsang, "What makes agile test artifacts useful? an activity-based quality model from a practitioners' perspective," in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '20, (New York, NY, USA), Association for Computing Machinery, 2020

[4]: J. Fischbach, J. Frattini, A. Spaans, M. Kummeth, A. Vogelsang, D. Mendez, and M. Unterkalmsteiner, "Automatic detection of causality in requirement artifacts: The cira approach," in *Requirements Engineering: Foundation for Software Quality* (F. Dalpiaz and P. Spoletini, eds.), (Cham), pp. 19–36, Springer International Publishing, 2021

[5]: J. Fischbach, J. Frattini, D. Mendez, M. Unterkalmsteiner, H. Femmer, and A. Vogelsang, "How do practitioners interpret conditionals in requirements?," in *Product-Focused Software Process Improvement* (L. Ardito, A. Jedlitschka, M. Morisio, and M. Torchiano, eds.), (Cham), pp. 85–102, Springer International Publishing, 2021

[6]: J. Fischbach, T. Springer, J. Frattini, H. Femmer, A. Vogelsang, and D. Mendez, "Fine-grained causality extraction from natural language requirements using recursive neural tensor networks," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 60–69, 2021

[7]: J. Frattini, J. Fischbach, D. Mendez, M. Unterkalmsteiner, A. Vogelsang, and K. Wnuk, "Causality in requirements artifacts: prevalence, detection, and impact," *Requirements Engineering*, Feb 2022

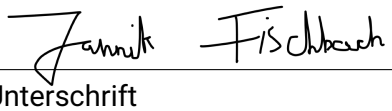
[8]: C. Wiecher, J. Fischbach, J. Greenyer, A. Vogelsang, C. Wolff, and R. Dumitrescu, "Integrated and iterative requirements analysis and test specification: A case study at kostal," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 112–122, 2021

[9]: N. Jadallah, J. Fischbach, J. Frattini, and A. Vogelsang, "Cate: Causality tree extractor from natural language requirements," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 77–79, 2021

[10]: J. Fischbach, J. Frattini, A. Vogelsang, D. Mendez, M. Unterkalmsteiner, A. Wehrle, P. R. Henao, P. Yousefi, T. Juricic, J. Radduenz, and C. Wiecher, "Automatic creation of acceptance tests by extracting conditionals from requirements: Nlp approach and case study," 2022

27.09.2022 — Datum

Jannik Fischbach — Name



Unterschrift