The Journal of Systems & Software 197 (2023) 111549

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

In Practice

Automatic creation of acceptance tests by extracting conditionals from requirements: NLP approach and case study[☆]

Jannik Fischbach[a,b,*], Julian Frattini[c], Andreas Vogelsang[d], Daniel Mendez[b,c], Michael Unterkalmsteiner[c], Andreas Wehrle[f], Pablo Restrepo Henao[a], Parisa Yousefi[e], Tedi Juricic[e], Jeannette Radduenz[f], Carsten Wiecher[g]

[a] *Netlight Consulting GmbH, Sternstraße 5, Munich, 80538, Germany*
[b] *fortiss GmbH, Guerickestraße 25, Munich, 80805, Germany*
[c] *Blekinge Institute of Technology, Valhallavägen 1, 371 41, Karlskrona, Sweden*
[d] *University of Cologne, Albertus-Magnus-Platz, 50923, Cologne, Germany*
[e] *Ericsson, Ölandsgatan 1, 371 33, Karlskrona, Sweden*
[f] *Allianz Deutschland AG, Dieselstr. 6, 85774, Unterföhring, Germany*
[g] *Leopold Kostal GmbH & Co. KG, An der Bellmerei 10, Lüdenscheid, 58513, Germany*

ARTICLE INFO

Article history:
Received 31 January 2022
Received in revised form 18 June 2022
Accepted 12 October 2022
Available online 20 October 2022

Dataset link: www.cira.bth.se/demo/, 10.5281/zenodo.5550387

Keywords:
Acceptance testing
Automatic test case creation
Requirements engineering
Natural language processing
Causality extraction

ABSTRACT

Acceptance testing is crucial to determine whether a system fulfills end-user requirements. However, the creation of acceptance tests is a laborious task entailing two major challenges: (1) practitioners need to determine the right set of test cases that fully covers a requirement, and (2) they need to create test cases manually due to insufficient tool support. Existing approaches for automatically deriving test cases require semi-formal or even formal notations of requirements, though unrestricted natural language is prevalent in practice. In this paper, we present our tool-supported approach CiRA (**C**onditionals **i**n **R**equirements **A**rtifacts) capable of creating the minimal set of required test cases from conditional statements in informal requirements. We demonstrate the feasibility of CiRA in a case study with three industry partners. In our study, out of 578 manually created test cases, 71.8 % can be generated automatically. Additionally, CiRA discovered 80 relevant test cases that were missed in manual test case design. CiRA is publicly available at www.cira.bth.se/demo/.

1. Introduction

Acceptance tests are used to verify the conformity between end-user requirements and actual system behavior (ISO/IEC/IEEE 24765:2010, E). Each acceptance test contains a finite set of test cases that specify certain test inputs and expected results. Test case design is a very laborious activity that easily accounts for 40 - 70 % of the total effort in the testing process (Beller et al., 2015). 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. A requirement is considered fully covered if a set of associated test cases assures the behavior implied by that requirement (Whalen et al., 2006). In a previous study (Fischbach et al., 2020a), 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.

Challenge 2 Creating acceptance tests is a predominantly manual task due to insufficient tool support (Garousi et al., 2020). Most of the existing approaches allow the derivation of test cases from semi-formal requirements (Wang et al., 2020; Carvalho et al., 2014; Barros et al., 2011) (e.g., expressed in controlled natural language) or formal requirements (Liu and Nakajima,

[☆] Editor: Marcos Kalinowski.

[*] Corresponding author at: Netlight Consulting GmbH, Sternstraße 5, Munich, 80538, Germany.

E-mail addresses: jannik.fischbach@netlight.com (J. Fischbach), julian.frattini@bth.se (J. Frattini), vogelsang@cs.uni-koeln.de (A. Vogelsang), daniel.mendez@bth.se (D. Mendez), michael.unterkalmsteiner@bth.se (M. Unterkalmsteiner), andreas.wehrle@allianz.de (A. Wehrle), pablo.restrepo@netlight.com (P.R. Henao), parisa.yousefi@ericsson.com (P. Yousefi), tedi.juricic@ericsson.com (T. Juricic), jeannette.radduenz@allianz.de (J. Radduenz), c.wiecher@kostal.com (C. Wiecher).

https://doi.org/10.1016/j.jss.2022.111549
0164-1212/© 2022 Published by Elsevier Inc.

2020; Sharma and Biswas, 2014) (e.g., expressed in linear temporal logic), but are not suitable to process informal requirements. However, studies (Kassab et al., 2014) have shown that requirements are usually expressed in unrestricted natural language (NL). Some approaches (Fischbach et al., 2020c; Verma and Beg, 2013; Santiago Júnior and Vijaykumar, 2012) address this research gap and focus on deriving test cases from informal requirements. Nevertheless, 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 (Fischbach et al., 2021c).

Research goal We aim to develop a tool-supported approach to derive the minimal set of required test cases automatically from NL requirements by applying *Natural Language Processing* (NLP).

Principal idea 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)". Previous studies (Fischbach et al., 2021b, 2020c) show that such conditional statements are prevalent in both traditional and agile requirements such as acceptance criteria. In this paper, we focus on conditionals in NL requirements and utilize their embedded logical knowledge for the automatic derivation of test cases. We answer three research questions (RQ):

- **RQ 1**: How to extract conditionals from NL requirements and use their implied relationships for automatic test case derivation?
- **RQ 2**: Can our automated approach create the same test cases as the manual approach?
- **RQ 3**: What are the reasons for deviating test cases?

The answers to RQ 1 shall inform the implementation of a new tool-supported approach for automatic test case derivation. RQ 2 and RQ 3 study the impact of the new 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 and compare automatically created test cases with existing, manually created test cases. In summary, this paper makes the following contributions (C):

- **C 1**: To answer RQ 1, we present our tool-supported approach CiRA (**C**onditionals **i**n **R**equirements **A**rtifacts) capable of (1) detecting conditional statements in NL requirements, (2) extracting their implied relationships in fine-grained form and (3) mapping these relationships to a *Cause–Effect–Graph* from which the minimal set of required test cases can be derived automatically. The output of CiRA are manual test cases, which – if required – can be converted into automatic test cases using third-party tools such as Selenium, Robot Framework, or Tricentis.
- **C 2**: To answer RQ 2 and RQ 3, we conduct a case study with three companies and compare CiRA to the manual test case design. We show that CiRA is able to automatically generate 71.8% of the 578 manually created test cases. In addition, CiRA identifies 80 relevant test cases that were missed in the manual test case design.
- **C 3**: To strengthen transparency and facilitate replication, we make our tool, code, annotated data set, and all trained models publicly available.[1]

- **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.

Fig. 1. Requirements specification of THEMAS (Ferrari et al., 2017).

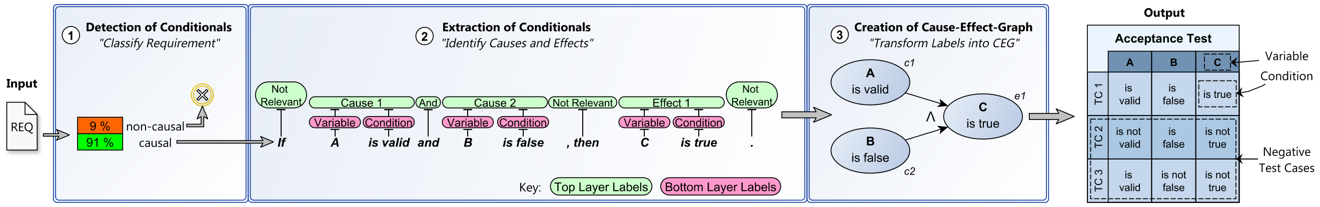
Running example In the course of the paper, we demonstrate the functionality of CiRA by means of a running example. Specifically, we explain how CiRA automatically derives the minimum number of required test cases for the requirements specification shown in Fig. 1. The specification contains an excerpt 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 (PUblic REquirements) data set (Ferrari et al., 2017) that contains 79 publicly available natural language requirements documents collected from the Web. We encourage the readers of this paper to use our online demo to process the running example on their own, allowing them to follow each individual step of CiRA.

Outline The remainder of this paper is organized as follows: Section 2 provides the theoretical background. Section 3 answers RQ 1 and introduces CiRA in detail. Section 4 presents the results of our case study and answers RQ 2 and RQ 3. Section 5 discusses our results and indicates directions for both research and practice. Section 6 briefly surveys related work. Finally, Section 7 presents our conclusions.

2 Fundamentals

Test case A test case is a set of certain test inputs (input parameters) and expected results (output parameters) used to verify compliance with a specific requirement (ISO/IEC/IEEE 24765:2010, E). Each input and output parameter is defined by a variable and a condition that the parameter can take (Sneed, 2007). For example, the parameter "the system detects an error" can be decomposed

[1] A demo of CiRA can be accessed at http://www.cira.bth.se/demo/. Our code, annotated data sets, and all trained models can be found at DOI10.5281/zenodo.5550387.



	A	B	C
TC 1	is valid	is false	is true
TC 2	is not valid	is false	is not true
TC 3	is valid	is not false	is not true

Fig. 2. Overview of the CiRA pipeline 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.*

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 (see Fig. 2, right). 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.

Conditional statements A conditional statement (short: conditional) 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*. This can be illustrated by the following REQ:

$$\text{If } \underbrace{\text{the system detects an error}}_{\text{antecedent}}, \underbrace{\text{an error message shall be shown}}_{\text{consequent}}.$$

The relationship between an antecedent and consequent can be interpreted logically in two different ways (Fischbach et al., 2020b). First, by means of an implication as $e_1 \Rightarrow e_2$, in which the antecedent is a *sufficient* condition for the consequent. Interpreting REQ as an implication requires the system to display an error message if the antecedent is true. However, it is not specified what the system should do if the antecedent is false. The implication allows both the occurrence of the consequent and its absence if the antecedent is false. The case of e_1 = false is *underspecified*. From a testing point of view, underspecified requirements can be problematic because the negative scenario is not defined. In fact, when reading REQ it may as well be reasonable to assume that the error message shall not be shown if the error has not been detected. This interpretation corresponds to a logical equivalence ($e_1 \Leftrightarrow e_2$), where the antecedent is both a *sufficient* and *necessary* condition for the consequent. Interpreting REQ as an equivalence requires the system to display an error message if and only if it detects an error.

In a previous study (Fischbach et al., 2021a), we found that conditionals in requirements are interpreted ambiguously and that practitioners disagree whether antecedents are only *sufficient* or also *necessary* for the consequent. To ensure that the automatically derived test cases correspond to the different logical interpretations, we require two variants of test case generation: 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. CiRA supports both variants of test case generation. The user can choose whether s/he perceives antecedents to be both *sufficient* and *necessary* conditions for consequents or not. 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 (see Fig. 2), we perceive the antecedents as *necessary* conditions for the consequent. Accordingly, our approach created one positive test case (see TC 1) and two negative test cases (see TC 2 and TC 3). For the sake of readability, in the remainder of the paper, we denote antecedents as causes and consequents as effects. Additionally, we term sentences that contain conditionals as causal sentences.

Cause Effect Graph A *Cause–Effect–Graph* (CEG) can be interpreted as a combinatorial logic network, which describes the interaction of causes and effects by Boolean logic (Nursimulu and Probert, 1995). It consists of nodes for each cause and effect and uses arcs with Boolean operators (conjunction $\wedge$, disjunction $\vee$, negation $\neg$) to illustrate the relationship between the nodes. Let G be the CEG shown in Fig. 2 with effect set E and cause set C. In the example, $|C| = 2$ including $c1$, and $c2$ while $|E| = 1$ with $e1$. To derive test cases from G, the *Basic Path Sensitization Technique* (BPST) is applied. The graph is traversed back from the effects to the causes and test cases are created according to specific decision rules (cf. Myers et al. (2012) and Nursimulu and Probert (1995)). 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. Hence, *Cause–Effect-Graphing* allows us to support practitioners in balancing between sufficient test coverage and the lowest possible number of test cases. This is also indicated by the acceptance test in Fig. 2, which contains the test cases generated by BPST for G. To check the functionality presented in the CEG comprehensively, only 3 test cases are needed instead of the maximum number of 2^3 test cases.

3 CiRA pipeline

As shown in Fig. 2, CiRA consists of three steps: We first detect whether an NL requirement contains a conditional (see Section 3.1). Second, we extract the conditional in fine-grained form (see Section 3.2). Specifically, we consider the combinatorics between causes and effects and split them into more granular text fragments (e.g., variable and condition), making the extracted conditionals suitable for automatic test case derivation. Third, we map the extracted causes and effects into a CEG to derive the minimum number of required test cases (see Section 3.3).

Input representation CiRA is not dependent on any specific format of requirements. Rather, it is able to process any kind of NL representation (i.e., CiRA does not dictate the semantics and syntax that a requirements author must follow). Hence, it understands unstructured representations (e.g., If event A occurs, then event B evaluates to true) as well as semi-structured formulations (e.g., IF A is true AND B is false THEN the system shall shut down). However, CiRA is not trained to process requirements that span multiple sentences.

3.1 Detection of conditionals

Problem We define the detection of conditionals in NL sentences as a binary classification problem, in which we are given a certain NL sentence $\mathcal{X}$ and we are required to produce a nominal label $y \in \mathcal{Y} = \{\text{causal, non-causal}\}$.

Novelty In previous work (Fischbach et al., 2021b), we compared the performance of different approaches for this task: baseline systems that search for cue phrases (e.g., *if*, *when*) that usually indicate causes and effects, Machine Learning (ML) approaches

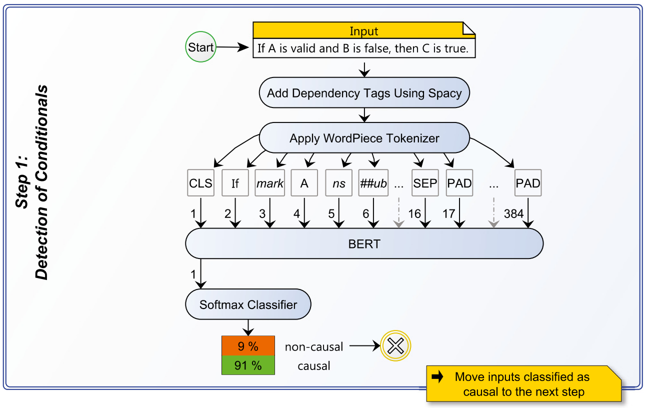

Step 1:
Detection of Conditionals
Start
Input
If A is valid and B is false, then C is true.
Add Dependency Tags Using Spacy
Apply WordPiece Tokenizer
CLS
If
mark
A
ns
##ub
...
SEP
PAD
...
PAD
1
2
3
4
5
6
16
17
384
BERT
1
Softmax Classifier
9 %
91 %
non-causal
causal
Move inputs classified as causal to the next step

Fig. 3. In-depth visualization of the first step in the CiRA pipeline: the detection of conditionals in NL requirements. Processed REQ: *If A is valid and B is false, then C is true.*

(e.g., Random Forest) and Deep Learning (DL) approaches (e.g., BERT+Softmax). In this paper, we present only the best-performing method that combines syntactically enriched BERT embeddings with a softmax classifier. For a detailed comparison of all investigated approaches, please refer to our manuscript (Fischbach et al., 2021b). Our detection approach is trained on 8,430 NL requirements of which 4,215 sentences are causal. It achieves a *macro*-F_1 score of 82 % and outperforms related approaches with an average gain of 11.06% in *macro*-Recall and 11.43% in *macro*-Precision.

Solution Our detection algorithm consists of two layers: (1) embedding layer and (2) inference layer. For a detailed visualization of the internal behavior of the individual layers, please refer to Fig. 3.

(1) Embedding Layer We represent each sentence as a sequence of word embeddings. Let us denote s as a sentence with n tokens: $s = \{v_1, v_2, \dots, v_n\}$, where vector v_i represents the vector of ith token with a dimension of d. In recent years, several methods have been developed to implement word embeddings. Traditional methods like word2vec (Mikolov et al., 2013) are capable of transforming a word into a single vector representation. However, they do not consider the context of the word. Hence, words are always represented as the same vector, although they can have different meanings depending on their context. To address this problem, contextual word embeddings like Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) were developed. Since BERT outperforms its predecessors, ELMo and GPT-2, in a number of NLP tasks, we use BERT embeddings in our proposed architecture. BERT requires input sequences with a fixed length. Therefore, sentences that are shorter are adjusted to this fixed length by appending padding tokens (PAD). Other tokens such as the separator (SEP) and classification (CLS) token, are also inserted to provide further information about the sentence to the model, where SEP marks the end of a sentence. 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). Studies (Sundararaman et al., 2019) have shown that the performance of NLP models can be improved by providing explicit prior knowledge of syntactic information to the model. Hence, we enrich the input sequence by adding the corresponding dependency (DEP) tag to each token and feed it into BERT. To choose a suitable fixed length for our input sequences, we analyzed the lengths of the sentences in our data set. A length of 384 tokens showed to be reasonable and allowed us to keep BERT's computational requirements to a minimum.

(2) Inference Layer For our classification task, we only 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: $\hat{y} = \text{softmax}(Wv_0 + b)$, where $\hat{y}$ is the predicted label for the sentence, W is the weighted matrix, v_0 is the first token in the sentence (i.e., the CLS token), and b is the bias. We select the class with the highest probability as the final classification result.

Running example When applying CiRA to the requirements specification shown in Fig. 1, 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 causal or not. In the present example, CiRA classifies REQ A, REQ B, REQ D, REQ E, REQ F, and REQ H as causal 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.

3.2 Extraction of conditionals

Problem 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 relevant for test case derivation 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 causes and effects (Fischbach et al., 2021b), we create individual labels for each cause and effect to clearly separate and map them easily to a CEG:

1. **Cause 1**
2. **Cause 2** } Cause Labels
3. **Cause 3**
4. **Effect 1**
5. **Effect 2** } Effect Labels
6. **Effect 3**
7. **Not Relevant** Marks parts of a conditional that are not relevant for automatic test derivation.
8. **And** Marks a conjunctive link between two adjacent causes or effects.
9. **Or** Marks a disjunctive link between two adjacent causes or effects.
10. **Variable** Marks the variable of a cause or effect.
11. **Condition** Marks the condition of a cause or effect.
12. **Negation**: Marks negated causes or effects.

We use these token labels to generate two annotation layers (see Fig. 2). The top layer represents the composition of the sentence by specifying the causes, effects, and their combinatorics based on the labels 1–9. At the lower layer, we use the labels 10–12 to annotate the causes and effects more fine-grained. Consequently, we assign at least one label and at most two labels to a token.

Novelty Contrary to the first part of CiRA (the detection of conditionals) we do not build on previous work for the fine-grained extraction of conditionals and present a new approach in this paper.

Table 1
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.

	Label type	Complete dataset			Training set			Validation set			Testing set			Annotation validity
		Sentences	WordPiece tokens	BPE tokens	Sentences	WordPiece tokens	BPE tokens	Sentences	WordPiece tokens	BPE tokens	Sentences	WordPiece tokens	BPE tokens	
Top layer	Cause 1	1946	18743	18317	1556	14862	14499	194	1878	1848	196	2003	1970	87 %
	Cause 2	661	5158	5190	523	4036	4072	68	540	534	70	582	584	71 %
	Cause 3	137	1109	1102	105	856	853	18	123	117	14	130	132	71 %
	Effect 1	1946	22814	22115	1556	18370	17832	194	2210	2125	196	2234	2158	90 %
	Effect 2	614	5384	5426	483	4169	4200	65	573	578	66	642	648	81 %
	Effect 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 %

3.2.1 Corpus creation

Data collection To train our extraction approach, we require an annotated data set, in which the combinatorics of causes and effects as well as their variables and conditions are labeled. Existing data sets (Xu et al., 2020) are not suitable for our use case: The SemEval-2007 (Girju et al., 2007) and SemEval-2010 (Hendrickx et al., 2010) data sets contain only single word causal pairs. In the data set presented by Dasgupta et al. (2018), causes and effects 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 by randomly selecting a subset of the causal requirements (1,946) and annotating 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. 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 (Stenetorp et al., 2012) for labeling each sentence.

Annotation validity To verify the reliability of the annotations, we calculate the inter-annotator agreement. We distribute the 1,946 causal sentences among four annotators, ensuring that 390 sentences are labeled by two annotators (overlapping quote of ≈20 %). Similar to other studies (Kolditz et al., 2019) that also utilize *brat* to annotate sentences, we calculate the pair-wise averaged F_1 score (Hripcsak, 2005) 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 1). The lowest agreement is achieved for **Cause 2** and **Cause 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 cause, or is it still part of the previous cause?). The highest agreement is measured for the assignment of **And** (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 1 shows that the majority of our sentences contain only a single cause and effect. About one-third of the sentences contain more complex conditionals comprising two causes or two effects. Only a few sentences contain three causes or three effects. We found that causes and effects are more often connected by a conjunction than by a disjunction. Negated causes and effects occur in about 18 % of the sentences. At the token level, expressions labeled as **Effect 1** are often longer than **Cause 1** expressions. We observe a similar trend at the lower layer. **Conditions** are usually longer than the **Variables** of the causes and effects. Across all classes, our data set is strongly unbalanced with four minority classes: **Cause 3**, **Effect 3**, **Or**, and **Negations**.

3.2.2 Solution

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 causes and effects, while the second model splits the causes and effects 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. The difference between our multi-class and multi-label solution can be illustrated using the token "A" shown in Fig. 2. In the case of our multi-class solution, the first model assigns the label **Cause 1** to the token, while the second model assigns the label **Variable** to the token. Conversely, in the case of our multi-label solution, both labels are assigned by a single model.

Models In our experiments, we compare the performance of nine different models for multi-class and multi-label classification. Table 2 provides an overview of their architectures consisting of three different layers: (1) embedding layer, (2) BiLSTM layer, and (3) inference layer.

(1) Embedding Layer Similar to the detection of conditionals, we also use contextual word embeddings for their extraction. However, we do not only perform experiments with BERT but also investigate the influence of RoBERTa (Liu et al., 2019) and DistilBERT (Sanh et al., 2019) embeddings on the performance of our models. DistilBERT represents the distilled version of BERT that allows for faster training. RoBERTa is a tuned version of BERT, which shows better prediction performance on various benchmarks but negatively affects training and inference time. To extract the conditionals in fine-grained form, we need to predict on the token level. Hence, in contrast to 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). We hypothesize that the performance degradation stems from the fact that the synthetic tokens contain valuable information about the syntax and semantics of a sentence helping the model to comprehend the meaning of a sentence. The number of tokens per class differs depending on the applied tokenizer (see Table 1). BERT and DistilBERT use WordPiece as a subword tokenization algorithm, while RoBERTA employs byte-pair encoding (BPE). Nevertheless, both tokenizers differ only slightly, so 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.

(2) Bidirectional-LSTM Layer For models VII - IX, we feed the word vectors into a Bidirectional LSTM (BiLSTM) to obtain a hidden state for each word. BiLSTMs 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: $h_i = \overrightarrow{h}_i \oplus \overleftarrow{h_i}$

Table 2
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.

	model #	Architecture			*Macro-F_1*	Optimal hyperparameters
		Embedding	BiLSTM	Inference		
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 = 9.47e-05, lstm_hidden = 128, d = 0.50

(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. Specifically, we use a softmax layer, which calculates the class probabilities for each token: $\hat{y} = \text{softmax}(Wv_i+b)$, where $\hat{y}$ are the predicted label probabilities for the ith token, W is the weighted matrix, and b is 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 Eq. (1)) while the lower layer is annotated with only three labels (see Eq. (2)). In essence, the first model aims to identify causes, effects, and their combinatorics by assigning **Cause 1–3**, **Effect 1–3**, and **Or/And** labels. The second model focuses on the decomposition of causes and effects by assigning **Variable** and **Condition** labels.

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

$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{3} exp(x_j)} \tag{2}$$

In case of the models IV - VI, we use a sigmoid layer to perform multi-label classification: $\hat{y} = \text{sigmoid}(Wv_i + b)$. 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: $\hat{y} = \text{sigmoid}(Wh_i + b)$

3.2.3 Experiments

Evaluation procedure We follow the idea of *Cross Validation* and 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 1). We opt for 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 (James et al., 2013). We use Precision, Recall, and F_1 score for evaluating our models. Since a single run of a k-fold cross-validation may result in a noisy estimate of model performance, we repeat the cross-validation procedure five times and average the scores from all repetitions.

Our data set is strongly unbalanced. Hence, we need to interpret the evaluation 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 class like **Cause 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.

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 for our models, we use the *Tree-structured Parzen Estimator* algorithm (Bergstra et al., 2011). 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 2 shows the best hyperparameters for each model.

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.

(1) 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 2). 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.

(2) 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 2). 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.

(3) 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 layer. We hypothesize that our amount of training instances is not adequate to sufficiently train the complex BiLSTM architecture and take advantage of its benefits.

(4) Label Prediction Table 3 indicates that model V is capable of processing both conditional statements consisting of only one cause as well as conditionals with multiple causes. Our model predicts **Cause 1** with very high Precision and Recall resulting in a *macro-*F_1 score of 91 %. For the prediction of **Cause 2** and **Cause 3**, our model also performs well by achieving *macro-*F_1 scores of 77 % and 82 %, respectively. Conditionals that contain only one effect or two effects can also be processed well by our model. However, our experiments show that the model lacks certainty in the prediction of **Effect 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 causes and effects. 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 causes and effects into more granular fragments. In addition, model V reliably identifies negations within the conditional statements. **(5) Summary** 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. A detailed insight into the functionality of model V is given in Fig. 4.

Running example In the second step, CiRA extracts the conditional statements from the requirements that were classified as causal 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 (**Cause 1**, **Cause 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. Fig. 6 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 cause and effect (see REQ D and REQ F) or complex conditionals with multiple causes and effects (see REQ B). Further, CiRA is able to detect causes and effects in different positions in a sentence, which can be illustrated by REQ E. In this case, CiRA extracts the conditional statement correctly even though **Cause 1** and **Cause 2** do not immediately follow each other but instead are separated by **Effect 1**.

Table 3
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	Cause 1	92 %	89 %	**91 %**
	Cause 2	83 %	72 %	77 %
	Cause 3	76 %	88 %	82 %
	Effect 1	90 %	89 %	**90 %**
	Effect 2	83 %	85 %	84 %
	Effect 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 %

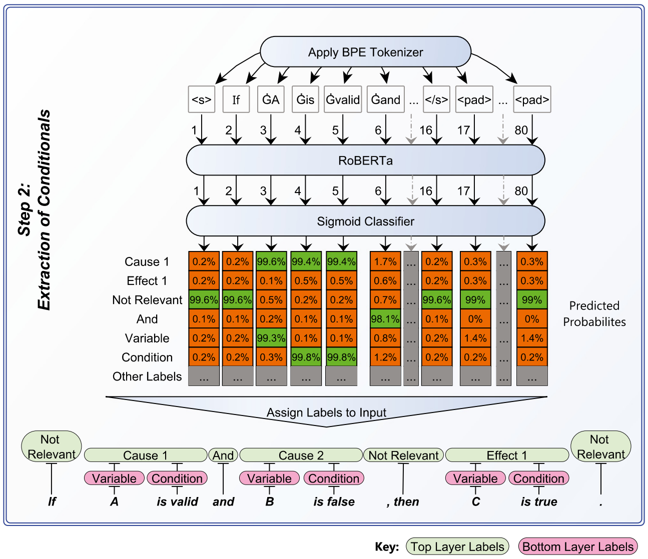


Fig. 4. In-depth visualization of the second step in the CiRA pipeline: the fine-grained extraction of conditionals in NL requirements. Processed REQ: *If A is valid and B is false, then C is true.*

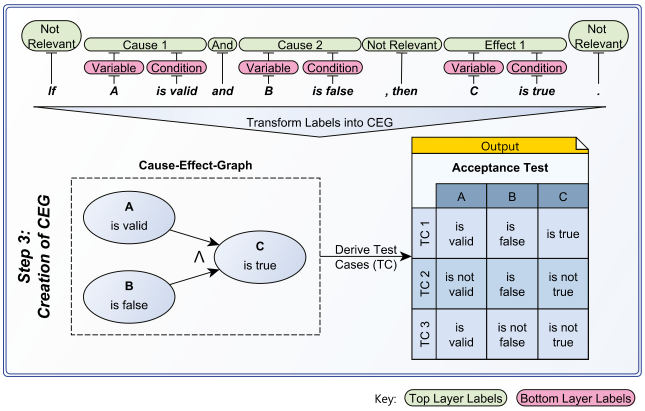
3.3 Creation of Cause–Effect–Graph

In the final step, we produce a CEG based on the extracted causes and effects. Creating a CEG is not a trivial task, especially for complex conditional statements consisting of multiple causes and effects. We handle the following cases:

❶ **Single Cause — Single Effect** In the simplest case, we create two nodes and draw an edge from the cause node to the effect node.

❷ **Multiple conjunctive Causes — Single Effect** In this case, all causes must occur jointly for the effect to occur. Thus, we connect all cause nodes with the effect node using the connective $\wedge$. We illustrate this case by using an exemplary requirement in Fig. 5.

❸ **Multiple disjunctive Causes — Single Effect** In this case, the occurrence of one of the causes is sufficient for the effect to



	A	B	C
TC 1	is valid	is false	is true
TC 2	is not valid	is false	is not true
TC 3	is valid	is not false	is not true

Fig. 5. In-depth visualization of the third step in the CiRA pipeline: the creation of a *Cause–Effect-Graph*. Processed REQ: *If A is valid and B is false, then C is true.*

occur. Thus, we link all cause nodes with the effect node using the logical connective ∨.

❹ **Combination of conjunctive and disjunctive Causes — Single Effect** 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 causes and connect the causes with the intermediate node using the logical connective ∧. Subsequently, we connect the disjunctive cause(s) and the intermediate node(s) with the effect using the logical connective ∨. If no connective can be found between two adjacent causes, 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.

❺ **Multiple conjunctive Effects** We create a node for each effect and connect them to the causes 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.

❻ **Correction of incomplete nodes** In the simplest case, a cause or effect 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.

Running example In the third step, CiRA interprets the conditional statements extracted from REQ A, REQ B, REQ D, REQ E, REQ F, and REQ H, and creates a corresponding CEG. Subsequently, CiRA applies the BPST (cf. Myers et al. (2012) and Nursimulu and Probert (1995)) to derive the minimum number of required test cases from each CEG. Fig. 6 presents an overview of the CEGs created by CiRA and the respective automatically generated test specifications for each causal requirement included in the THEMAS requirements specification (see Fig. 1).

The CEG generated for REQ A corresponds to case ❺ described above. Specifically, one cause is the trigger for two conjunctive effects: "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 **Effect 2** is not explicitly defined in the requirement. We must therefore automatically complete the node of **Effect 2** by adopting the variable from **Effect 1** (see case ❻). REQ B specifies complex system behavior and contains two causes and two effects. When creating a corresponding CEG, we need to consider both case ❸ and case ❺. Thus, we link all cause nodes with both effect nodes using the logical connective ∨. Similar to REQ A, we need to automatically complete the variable of **Effect 2** since it is not explicitly defined in REQ B (see case ❻). REQ D contains a negated cause that is responsible for the occurrence of a single effect. In other words, not exceeding the limit is required for a temperature to be eligible for further processing. Therefore, we are dealing with case ❶ and have to negate the edge between cause and effect. Once again, the variable of an effect node is not described in the requirement. In contrast to REQ A and REQ B we do not complement the node of **Effect 1** with the variable of a neighboring effect but rather adopt the variable of **Cause 1**. The CEG generated for REQ E corresponds to case ❷. Hence, we connect both cause nodes with the single effect using the connective ∧. REQ F includes a negated effect that is triggered by a single cause. We thus create the CEG based on the rules described in case ❶ and negate the edge between cause and effect. REQ H contains one cause and two conjunctive effects. We create a corresponding CEG by applying the rules described in case ❺. We complement the node of **Effect 2** by adopting the variable of **Effect 1**.

CiRA automatically created a total of 14 test cases for all causal requirements 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. 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.

Answer to RQ 1:

We develop CiRA, an approach to identify and extract conditionals in NL and transfer them into a CEG, from which the minimal set of required test cases can be derived automatically. For the detection, we use syntactically enriched BERT embeddings combined with a softmax classifier (Fischbach et al., 2021b). Our experiments show that a sigmoid classifier built on RoBERTA embeddings (see model V*) is best suited to extract conditionals in fine-grained form.*

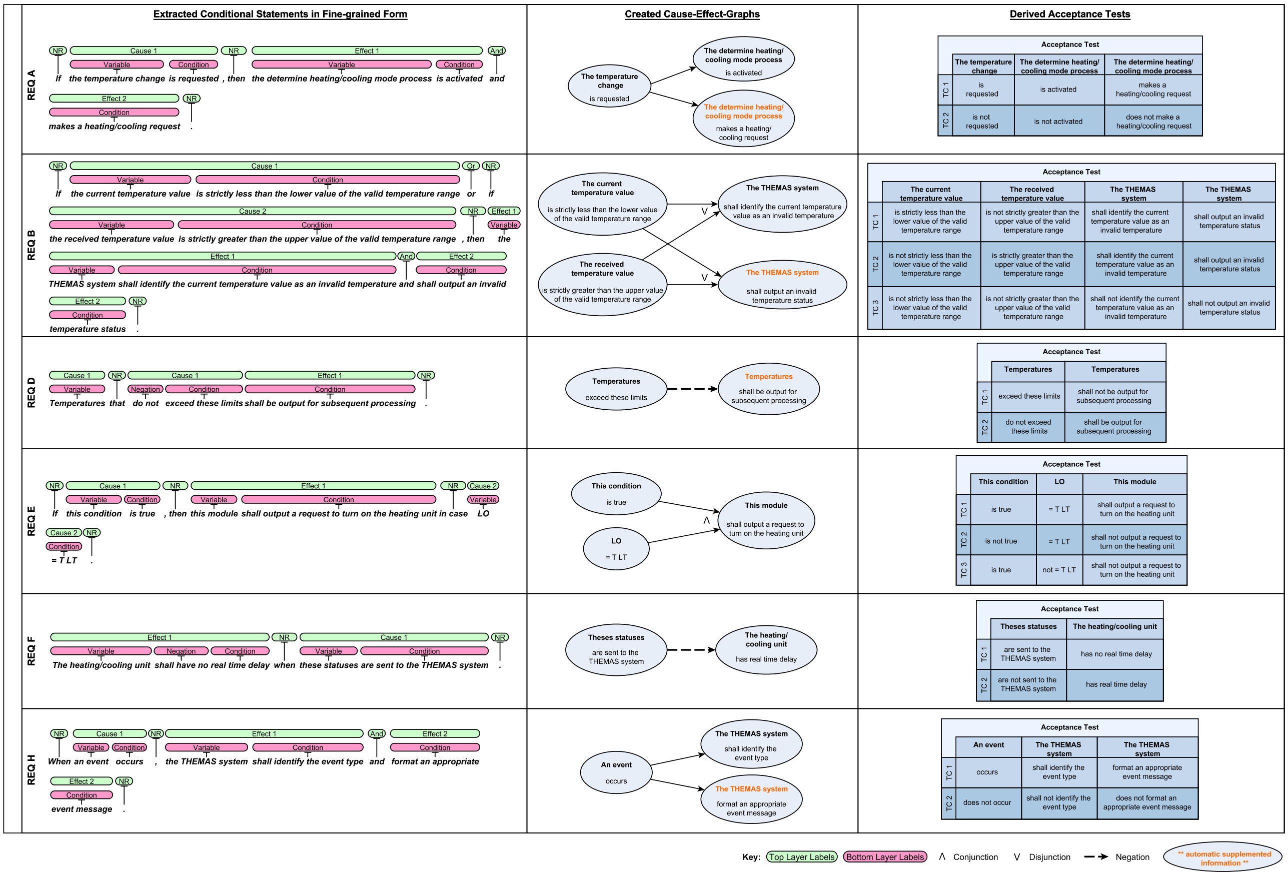
4 Case study: CiRA in practice

Objective To answer RQ 2 and RQ 3, we conduct a case study in an exploratory fashion. We aim to evaluate whether CiRA could either replace or augment the existing manual approach for creating test cases. For our study, we follow the guidelines by Runeson and Höst (2009) for conducting case study research.

4.1 Case sampling and study objects

We apply purposive case sampling augmented with convenience sampling (Kitchenham and Pfleeger, 2002). 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 acceptance criteria describing the functionality of a business information system used for vehicle


Extracted Conditional Statements in Fine-grained Form
Created Cause-Effect-Graphs
Derived Acceptance Tests
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 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 H
When an event occurs , the THEMAS system shall identify the event type and format an appropriate event message .
NR
Cause 1
Cause 2
Effect 1
Effect 2
And
Or
Variable
Condition
Negation
The temperature change is requested
The determine heating/cooling mode process is activated
The determine heating/cooling mode process makes a heating/cooling request
The current temperature value is strictly less than the lower value of the valid temperature range
The received temperature value is strictly greater than the upper value of the valid temperature range
The THEMAS system shall identify the current temperature value as an invalid temperature
The THEMAS system shall output an invalid temperature status
Temperatures exceed these limits
Temperatures shall be output for subsequent processing
This condition is true
LO = T LT
This module shall output a request to turn on the heating unit
Theses statuses are sent to the THEMAS system
The heating/cooling unit has real time delay
An event occurs
The THEMAS system shall identify the event type
The THEMAS system format an appropriate event message
Key: Top Layer Labels Bottom Layer Labels Λ Conjunction V Disjunction Negation
** automatic supplemented information **

Acceptance Test

	The temperature change	The determine heating/ cooling mode process	The determine heating/ cooling mode process
TC 1	is requested	is activated	makes a heating/cooling request
TC 2	is not requested	is not activated	does not make a heating/cooling request

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

Acceptance Test

	Temperatures	Temperatures
TC 1	exceed these limits	shall not be output for subsequent processing
TC 2	do not exceed these limits	shall be output for subsequent processing

Acceptance Test

	This condition	LO	This module
TC 1	is true	= T LT	shall output a request to turn on the heating unit
TC 2	is not true	= T LT	shall not output a request to turn on the heating unit
TC 3	is true	not = T LT	shall not output a request to turn on the heating unit

Acceptance Test

	Theses statuses	The heating/cooling unit
TC 1	are sent to the THEMAS system	has no real time delay
TC 2	are not sent to the THEMAS system	has real time delay

Acceptance Test

	An event	The THEMAS system	The THEMAS system
TC 1	occurs	shall identify the event type	format an appropriate event message
TC 2	does not occur	shall not identify the event type	does not format an appropriate event message

Fig. 6. Overview of the conditionals extracted by CiRA in fine grained form (left), the generated *Cause–Effect-Graphs* (center), and the derived acceptance tests (right) per requirement defined in the THEMAS specification.

insurance. 127 of these acceptance criteria follow a causal pattern and are therefore suitable for assessing CiRA. The remaining acceptance criteria 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 causal acceptance criteria. 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 causal requirements, which corresponds to about 1.33 test cases per acceptance test.

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 are indeed causal 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 causal requirements. In total, 204 test cases were designed, which corresponds to about 2.58 test cases per acceptance test.

4.2 Study design

Approach for RQ 2 We want to study whether CiRA 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 CiRA 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 an 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:

- *Identical*: A test case that has been created manually as well as automatically by CiRA.

- $\boxed{AA \wedge rel}$: A test case that has been missed in manual test design and should be included in the acceptance test.
- $\boxed{AA \wedge \neg rel}$: A superfluous test case that is correctly not included in the manually created acceptance test.
- $\boxed{MA \wedge rel}$: A test case that has been missed by CiRA and should be included in the acceptance test.
- $\boxed{MA \wedge \neg rel}$: A superfluous test case that is correctly not included in the automatically created acceptance test.

Approach for RQ 3 To answer the third RQ, we document all errors and ambiguities of CiRA produced during the approach for RQ2 to discuss these with 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 interview one or more 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: a meeting between one of the two first authors and each respective company is scheduled during which all recorded deviations are presented, jointly discussed and their final categorization according to the aforementioned five categories is confirmed. The first author involves two test designers at Allianz and two test designers at Kostal, the second author involves one test designer at Ericsson.

4.3 Case analysis

We report on our findings structured by our research questions (RQ 2 and RQ 3). Fig. 7 provides an overview of manually and automatically created test cases.

RQ 2: Can CiRA create the same test cases as the manual approach?
Findings at Allianz CiRA detected 90.55% of the causal acceptance criteria. Consequently, no test cases were created for the missed 12 causal criteria. For the correctly classified criteria, CiRA 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 CiRA, of which 27 test cases are related to criteria that were incorrectly identified as non-causal. 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 causal requirements but failed to do so for ten causal requirements. 91 test cases were automatically generated based on these identified requirements, which corresponds to about 2.33 test cases per acceptance test. 28 manual test cases were automatically created by CiRA 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 CiRA, of which seven test cases are related to criteria that were incorrectly identified as non-causal. 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.

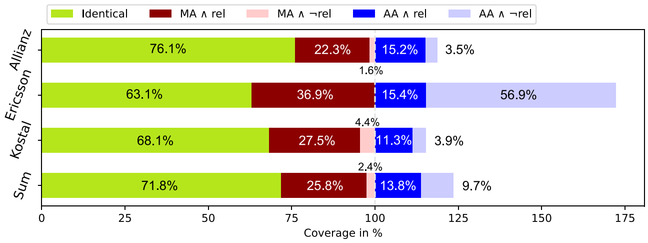


Fig. 7. Case study results. Comparison of manually and automatically created test cases.

Findings at Kostal CiRA correctly classified 72 requirements as causal. However, it failed to identify the remaining seven causal requirements. 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 causal 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 are correctly not included in the manual set.

Answer to RQ 2:

Across all case companies, CiRA automatically created 71.8 % of the 578 manually created test cases. CiRA 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 CiRA is able to automatically create a significant amount of relevant (known and new) test cases.

RQ 3: 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 CiRA could not generate 37 $\boxed{MA \wedge rel}$ test cases due to a lack of information in these requirements. At Allianz, 16 out of 127 causal acceptance

criteria lack information. Our analysis shows that CiRA could not generate 31 $\boxed{MA \wedge rel}$ test cases due to incomplete acceptance criteria. At Ericsson, 17 $\boxed{MA \wedge rel}$ 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 $\boxed{AA \wedge rel}$ test cases were actually negative cases. Only two positive cases were overlooked in the manual process. At Allianz, 36 of the 47 $\boxed{AA \wedge rel}$ test cases were actually negative cases. 11 positive cases were missed by the test designers. In the case of Ericsson, all ten $\boxed{AA \wedge rel}$ test cases were overlooked negative test cases.

Infeasible test cases Our analysis shows that some of the exclusively automatically created test cases cannot 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 cannot 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 $\boxed{AA \wedge \neg rel}$ test cases cannot be checked in practice. At Allianz, five of the eleven $\boxed{AA \wedge \neg rel}$ test cases can only occur theoretically. At Ericsson, 28 of 37 $\boxed{AA \wedge \neg rel}$ test cases fell into this category.

Cira errors CiRA 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, CiRA failed to generate 3 $\boxed{MA \wedge rel}$ test cases and instead created five $\boxed{AA \wedge \neg rel}$ test cases, because the generated CEG reflected a wrong causal relation. In the case of Allianz, CiRA failed to create eleven $\boxed{MA \wedge rel}$ test cases and instead generated six $\boxed{AA \wedge \neg rel}$ test cases. In the case of Ericsson, CiRA produced nine $\boxed{AA \wedge \neg rel}$ test cases due to incorrect interpretation of the causal relation. We found that these errors occurred mainly when the conditionals contained three or more effects. This confirms the findings from our experiment that CiRA struggles in reliably identifying more than two effects (see Table 3).

Answer to RQ 3:

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

5 Discussion

This section discusses our results and summarizes both the potentials as well as limitations of CiRA. Based on our discussion, we deduce key take-aways for practitioners.

5.1 Potentials, limitations, and key take-aways

Our case study demonstrates that CiRA is able to support practitioners in deriving relevant test cases from conditionals. Across all industry partners, CiRA automatically generates more than 70 % of the manually created test cases. However, CiRA does not achieve full automation of acceptance test creation, mainly due to incomplete requirements. CiRA 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 Fernández et al. (2017) that incompleteness is still a major problem in practice and hinders the automatic processing of requirements.

1. Key Take-away:

In fact, CiRA can help to generate acceptance tests automatically. However, CiRA 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 CiRA 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 CiRA 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 CiRA as a supplement to manual test design, test designers need to filter the automatically generated test cases. We however argue that this does not impede the usability of CiRA, as it is significantly easier to manually discard irrelevant test cases than to manually identify undetected, relevant test cases. We favor Recall over Precision since the performance gained due to the automatization outweighs the manual effort to filter the resulting set of proposed test cases.

2. Key Take-away:

CiRA 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 CiRA are necessarily relevant, requiring subsequent manual review of the automatically created test specifications.

Previous approaches (Fischbach et al., 2020c; Sorgente et al., 2013) for extracting conditionals analyze the grammatical structure of a sentence by dependency parsing and extract the conditionals from the dependency tree. However, dependency trees usually contain incorrect arcs for sentences that contain grammar mistakes causing the approaches to extract wrong conditionals or even to fail completely. 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 out-of-vocabulary words. Nevertheless, studies (Kumar et al., 2020) reveal that language models such as BERT show significant performance degradation with increasing amounts of noisy data. As a result, we hypothesize that CiRA's robustness against grammatical mistakes is limited to a few errors in a sentence. We, therefore, propose to combine CiRA with requirements smell checkers (Femmer et al., 2017) 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 causality and is not able to extract conditional statements that span multiple sentences. However, two-sentence causality may arise in practice (e.g., indicated by "therefore", "hence"), requiring us to extend CiRA in future work. According to the test designers, a further challenge in the extraction of conditionals relates to the handling of *event chains* (i.e., linked causal requirements, in which the effect of a conditional represents a cause in another conditional). In such cases, it is no longer sufficient to create a single CEG. Rather, we must create several CEGs and connect them to each other. Currently, CiRA only allows the creation of acceptance tests for causal requirements. 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 single causal requirements. 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 (Tran et al., 2021). 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.

5.2 Threats to validity

As in every empirical study, our case study (see RQ 2 and RQ 3) is also subject to potential validity threats. This section discusses these threats and describes how we mitigated them.

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 CiRA's performance.

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 CiRA may also be valuable for other companies considering that conditionals are widely used in requirements. Validation of this claim requires further empirical investigation.

6 Related work

Since the early 1980s, NLP techniques have been applied to RE artifacts to support a variety of use cases: e.g., requirements classification (Hey et al., 2020), topic modeling (Gülle et al., 2020), and quality checks (Femmer et al., 2017). A comprehensive overview of existing NLP4RE tools is provided by Zhao et al. (2021). In this paper, we use NLP methods to extract conditionals from requirements in fine-grained form and to derive test cases automatically. This section reviews existing approaches for both use cases and relates them to our work.

Approaches to test case derivation There is a rich body of work on automatically deriving test cases from semi-formal and formal requirements (Ahsan et al., 2017). Wang et al. (2020) and Zhang et al. (2014) describe how to automatically generate executable system test cases from restricted use case models. Sarmiento et al. (2016) explain how semi-formal requirements can be translated into Petri-Net models, which can be used as input for test scenario generation. Carvalho et al. (2014) present their tool NAT2TEST$_{SCR}$ and show how to automatically generate test cases from requirements written in SysReq-CNL. However, only a few approaches allow to automatically create test cases from informal requirements. Santiago Júnior and Vijaykumar (2012) present SOLIMVA capable of translating NL requirements into state charts used for the eventual test case generation. Verma and Beg (2013) describe a similar approach for translating informal requirements into knowledge representation graphs. Dwarakanath and Sengupta (2012) present Litmus, a tool that applies a syntactic parser called *Link Grammar* in order to analyze the structure of an NL requirement and create test cases accordingly. Goffi et al. (2016) demonstrate how to create test cases for exceptional behaviors from Javadoc comments. The mentioned approaches have three major drawbacks: (1) they require manual work such as the creation of a dictionary (Santiago Júnior and Vijaykumar, 2012) defining the application domain in which the approach will be used, (2) they do not ensure that only the minimal number of required test cases is created, and (3) they lack tool-support and are thus not immediately usable for practitioners. In previous work (Fischbach et al., 2020c), we address this research gap and present *Specmate* that automatically converts acceptance criteria into test cases by extracting cause–effect-relations based on dependency parsing. However, we found that *Specmate* is not robust against grammatical errors and fails to process words that are not yet part of its training vocabulary. In this paper, we therefore shift from using dependency parsing to solving the extraction of conditionals as a sequence labeling problem.

Approaches to extract conditionals Several approaches for conditional extraction have been developed (Yang et al., 2021): Rule-based approaches (Khoo et al., 1998; Puente and Olivas, 2008) extract conditionals by applying linguistic patterns such as *[cause] and because of this, [effect]*. Their performance relies on hand-coded patterns, which require extensive manual work. Other approaches use ML. Chang and Choi (2005) use a Naive Bayes classifier to predict the probability of a causal relation given a certain cue phrase (e.g., causative verb). Rink and Harabagiu (2010) propose a Support Vector Machines classifier trained on contextual features. More recent approaches (Dasgupta et al., 2018; Li et al., 2021) apply DL to automatically extract useful features from raw text. However, the existing approaches are not capable of extracting conditionals in fine-grained form, rendering them unsuitable for our use case. Specifically, some approaches extract only causal pairs or detect causal relations on the phrase level, but do not consider the combinatorics between causes and effects. In addition, they do not allow to split causes and effects into more granular text fragments (e.g., variable and condition). We addressed this research gap in previous work (Fischbach et al., 2021c) and trained a Recursive Neural Tensor Network (RNTN) to recover the composition of conditionals as binary trees. However, we found that the RNTN struggles to understand the semantics of out-of-vocabulary words posing a threat to its applicability in practice. Contrary, CiRA is able to handle unseen data due to subword tokenization.

7 Conclusion

Acceptance testing evaluates the conformance between actual and expected system behavior. However, the creation of acceptance tests is laborious and requires manual work due to missing tool support. In this paper, we focus on conditional statements in functional requirements and demonstrate how NLP can be used to automatically generate the minimum set of required test cases. Specifically, we present our tool-supported approach CiRA capable of (1) detecting conditional statements, (2) extracting them in fine-grained form, and (3) translating them into a CEG, from which we derive the minimal number of required test cases. We evaluate CiRA by conducting a case study with three companies. Our study demonstrates that CiRA is able to automatically create 71.8% of the 578 manually created test cases. Additionally, CiRA identified 80 relevant test cases that were missed in manual test design. Future research will both explore the integration of CiRA in an industrial context, but also explore how the CiRA approach can support further use cases. By providing alternative representations of causal requirements – both in the form of a CEG and a test suite – potential improvements to the understandability of requirements can be investigated. For example, CiRA could be used to create an immediate feedback loop to confirm that these alternative representations reflect the original intention of the possibly ambiguous natural language requirement.

CRediT authorship contribution statement

Jannik Fischbach: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision. **Julian Frattini:** Methodology, Software, Validation, Investigation, Data curation, Writing – review & editing, Visualization. **Andreas Vogelsang:** Conceptualization, Supervision, Project administration. **Daniel Mendez:** Methodology, Writing – review & editing, Supervision. **Michael Unterkalmsteiner:** Methodology, Writing – review & editing, Supervision. **Andreas Wehrle:** Data curation, Software. **Pablo Restrepo Henao:** Software. **Parisa Yousefi:** Resources, Investigation. **Tedi Juricic:** Resources, Investigation. **Jeannette Radduenz:** Resources, Investigation. **Carsten Wiecher:** Resources, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A live demo of CiRA can be accessed at www.cira.bth.se/demo/. Our code, annotated data sets, and all trained models are publicly available at 10.5281/zenodo.5550387.

Acknowledgments

We would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology. We thank all collaborating companies for providing access to their data and test designers who volunteered their time to participate in this research.

References

Ahsan, I., Butt, W.H., Ahmed, M.A., Anwar, M.W., 2017. 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, Association for Computing Machinery, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3018896.3036375.

Barros, F.A., Neves, L., Hori, E., Torres, D., 2011. The ucsCNL: A controlled natural language for use case specifications. In: Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE). pp. 250–253.

Beller, M., Gousios, G., Panichella, A., Zaidman, A., 2015. When, how, and why developers (do not) test in their IDEs. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. In: ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, pp. 179–190. http://dx.doi.org/10.1145/2786805.2786843.

Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B., 2011. Algorithms for hyper-parameter optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems. NIPS '11, Curran Associates Inc., Red Hook, NY, USA, pp. 2546–2554.

Carvalho, G., Barros, F., Lapschies, F., Schulze, U., Peleska, J., 2014. Model-based testing from controlled natural language requirements. In: Artho, C., Ölveczky, P.C. (Eds.), Formal Techniques for Safety-Critical Systems. Springer International Publishing, Cham, pp. 19–35.

Chang, D.-S., Choi, K.-S., 2005. Causal relation extraction using cue phrase and lexical pair probabilities. In: Su, K.-Y., Tsujii, J., Lee, J.-H., Kwong, O.Y. (Eds.), Natural Language Processing – IJCNLP 2004. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 61–70.

Dasgupta, T., Saha, R., Dey, L., Naskar, A., 2018. 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. Association for Computational Linguistics, Melbourne, Australia, pp. 306–316. http://dx.doi.org/10.18653/v1/W18-5035, URL https://aclanthology.org/W18-5035.

Devlin, J., Chang, M.-W., Lee, K., Toutanova, K., 2019. 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, Volume 1 (Long and Short Papers). Association for Computational Linguistics, Minneapolis, Minnesota, pp. 4171–4186. http://dx.doi.org/10.18653/v1/N19-1423, URL https://aclanthology.org/N19-1423.

Dwarakanath, A., Sengupta, S., 2012. Litmus: Generation of test cases from functional requirements in natural language. In: Bouma, G., Ittoo, A., Métais, E., Wortmann, H. (Eds.), Natural Language Processing and Information Systems. Springer, Berlin, Heidelberg, pp. 58–69.

Femmer, H., Méndez Fernández, D., Wagner, S., Eder, S., 2017. Rapid quality assurance with requirements smells. J. Syst. Softw. 123, 190–213. http://dx.doi.org/10.1016/j.jss.2016.02.047, URL https://www.sciencedirect.com/science/article/pii/S0164121216000789.

Fernández, D.M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.T., Greer, D., Lassenius, C., Männistö, T., Nayabi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., Spinola, R., Tuzcu, A., De La Vara, J.L., Wieringa, R., 2017. Naming the pain in requirements engineering. Empir. Softw. Eng. 22 (5), 2298–2338.

Ferrari, A., Spagnolo, G.O., Gnesi, S., 2017. PURE: A dataset of public requirements documents. In: 2017 IEEE 25th International Requirements Engineering Conference. RE, pp. 502–505. http://dx.doi.org/10.1109/RE.2017.29.

Fischbach, J., Femmer, H., Mendez, D., Fucci, D., Vogelsang, A., 2020a. 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, Association for Computing Machinery, New York, NY, USA, pp. 1–10. http://dx.doi.org/10.1145/3382494.3421462.

Fischbach, J., Frattini, J., Mendez, D., Unterkalmsteiner, M., Femmer, H., Vogelsang, A., 2021a. How do practitioners interpret conditionals in requirements? In: Ardito, L., Jedlitschka, A., Morisio, M., Torchiano, M. (Eds.), Product-Focused Software Process Improvement. Springer International Publishing, Cham, pp. 85–102.

Fischbach, J., Frattini, J., Spaans, A., Kummeth, M., Vogelsang, A., Mendez, D., Unterkalmsteiner, M., 2021b. Automatic detection of causality in requirement artifacts: The CiRA approach. In: Dalpiaz, F., Spoletini, P. (Eds.), Requirements Engineering: Foundation for Software Quality. Springer International Publishing, Cham, pp. 19–36.

Fischbach, J., Hauptmann, B., Konwitschny, L., Spies, D., Vogelsang, A., 2020b. Towards causality extraction from requirements. In: 2020 IEEE 28th International Requirements Engineering Conference. RE, pp. 388–393. http://dx.doi.org/10.1109/RE48521.2020.00053.

Fischbach, J., Springer, T., Frattini, J., Femmer, H., Vogelsang, A., Mendez, D., 2021c. 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. http://dx.doi.org/10.1109/REW53955.2021.00016.

Fischbach, J., Vogelsang, A., Spies, D., Wehrle, A., Junker, M., Freudenstein, D., 2020c. 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. http://dx.doi.org/10.1109/ICST46399.2020.00040.

Garousi, V., Bauer, S., Felderer, M., 2020. NLP-assisted software testing: A systematic mapping of the literature. Inf. Softw. Technol. 126, 106321. http://dx.doi.org/10.1016/j.infsof.2020.106321, URL https://www.sciencedirect.com/science/article/pii/S0950584920300744.

Girju, R., Nakov, P., Nastase, V., Szpakowicz, S., Turney, P., Yuret, D., 2007. Semeval-2007 task 04: Classification of semantic relations between nominals. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEVal-2007). Association for Computational Linguistics, Prague, Czech Republic, pp. 13–18, URL https://aclanthology.org/S07-1003.

Goffi, A., Gorla, A., Ernst, M.D., Pezzè, M., 2016. Automatic generation of oracles for exceptional behaviors. In: Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, http://dx.doi.org/10.1145/2931037.2931061.

Gülle, K.J., Ford, N., Ebel, P., Brokhausen, F., Vogelsang, A., 2020. Topic modeling on user stories using word mover's distance. In: 2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering. AIRE, pp. 52–60. http://dx.doi.org/10.1109/AIRE51212.2020.00015.

Hendrickx, I., Kim, S.N., Kozareva, Z., Nakov, P., Ó Séaghdha, D., Padó, S., Pennacchiotti, M., Romano, L., Szpakowicz, S., 2010. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In: Proceedings of the 5th International Workshop on Semantic Evaluation. Association for Computational Linguistics, Uppsala, Sweden, pp. 33–38, URL https://aclanthology.org/S10-1006.

Hey, T., Keim, J., Koziolek, A., Tichy, W.F., 2020. Norbert: Transfer learning for requirements classification. In: 2020 IEEE 28th International Requirements Engineering Conference. RE, pp. 169–179. http://dx.doi.org/10.1109/RE48521.2020.00028.

Hripcsak, G., 2005. Agreement, the F-measure, and reliability in information retrieval. J. Am. Med. Inform. Assoc. 12 (3), 296–298. http://dx.doi.org/10.1197/jamia.m1733.

ISO/IEC/IEEE 24765:2010(E), 2011. Systems and software engineering — Vocabulary. International Organization for Standardization, Geneva, CH.

James, G., Witten, D., Hastie, T., Tibshirani, R.E., 2013. An Introduction to Statistical Learning. vol. 112, Springer.

Kassab, M., Neill, C., Laplante, P., 2014. State of practice in requirements engineering: contemporary data. Innov. Syst. Softw. Eng. 10 (4), 235–241. http://dx.doi.org/10.1007/s11334-014-0232-4.

Khoo, C.S.G., Kornfilt, J., Oddy, R.N., Myaeng, S.H., 1998. Automatic extraction of cause-effect information from newspaper text without knowledge-based inferencing. Lit. Linguist. Comput. 13 (4), 177–186. http://dx.doi.org/10.1093/llc/13.4.177.

Kitchenham, B., Pfleeger, S.L., 2002. Principles of survey research: Part 5: Populations and samples. SIGSOFT Softw. Eng. Notes 27 (5), 17–20. http://dx.doi.org/10.1145/571681.571686.

Kolditz, T., Lohr, C., Hellrich, J., Modersohn, L., Betz, B., Kiehntopf, M., Hahn, U., 2019. Annotating german clinical documents for de-identification. Stud. Health Technol. Inform. 264, 203—207. http://dx.doi.org/10.3233/shti190212.

Kumar, A., Makhija, P., Gupta, A., 2020. Noisy text data: Achilles' heel of BERT. In: Proceedings of the Sixth Workshop on Noisy User-Generated Text (W-NUT 2020). Association for Computational Linguistics, pp. 16–21. http://dx.doi.org/10.18653/v1/2020.wnut-1.3, URL https://aclanthology.org/2020.wnut-1.3, (Online).

Li, Z., Li, Q., Zou, X., Ren, J., 2021. Causality extraction based on self-attentive BiLSTM-CRF with transferred embeddings. Neurocomputing 423, 207–219. http://dx.doi.org/10.1016/j.neucom.2020.08.078.

Liu, S., Nakajima, S., 2020. Automatic test case and test oracle generation based on functional scenarios in formal specifications for conformance testing. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2020.2999884.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. RoBERTa: A robustly optimized BERT pretraining approach. CoRR, abs/1907.11692.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. NIPS '13, Curran Associates Inc., Red Hook, NY, USA, pp. 3111–3119.

Myers, G.J., Badgett, T., Sandler, C. (Eds.), 2012. The art of software testing. Wiley, http://dx.doi.org/10.1002/9781119202486.

Nursimulu, K., Probert, R.L., 1995. 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, IBM Press, p. 46.

Puente, C., Olivas, J., 2008. Analysis, detection and classification of certain conditional sentences in text documents. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. pp. 1097–1104.

Rink, B., Harabagiu, S., 2010. UTD: Classifying semantic relations by combining lexical and semantic resources. In: Proceedings of the 5th International Workshop on Semantic Evaluation. Association for Computational Linguistics, Uppsala, Sweden, pp. 256–259, URL https://aclanthology.org/S10-1057.

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164.

Sanh, V., Debut, L., Chaumond, J., Wolf, T., 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. CoRR, abs/1910.01108.

Santiago Júnior, V.A.d., Vijaykumar, V.L., 2012. Generating model-based test cases from natural language requirements for space application software. Softw. Qual. J. 20 (1), 77–143.

Sarmiento, E., Leite, J.C., Almentero, E., Sotomayor Alzamora, G., 2016. Test scenario generation from natural language requirements descriptions based on Petri-nets. Electron. Notes Theor. Comput. Sci. 329, 123–148.

Sharma, R., Biswas, K.K., 2014. Automated generation of test cases from logical specification of software requirements. In: Filipe, J., Maciaszek, L.A. (Eds.), ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering, Lisbon, Portugal, 28-30 April, 2014. SciTePress, pp. 241–248. http://dx.doi.org/10.5220/0004972902410248.

Sneed, H.M., 2007. Testing against natural language requirements. In: 2007 7th International Conference on Quality Software. IEEE Computer Society, Los Alamitos, CA, USA, pp. 380–387. http://dx.doi.org/10.1109/QSIC.2007.61, URL https://doi.ieeecomputersociety.org/10.1109/QSIC.2007.61.

Sorgente, A., Vettigli, G., Mele, F., 2013. Automatic extraction of cause-effect relations in natural language text. In: DART@AI*IA. pp. 37–48.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., Tsujii, J., 2012. Brat: a web-based tool for NLP-assisted text annotation. In: Proceedings of the Demonstrations At the 13th Conference of the European Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, Avignon, France, pp. 102–107, URL https://aclanthology.org/E12-2021.

Sundararaman, D., Subramanian, V., Wang, G., Si, S., Shen, D., Wang, D., Carin, L., 2019. Syntax-infused transformer and BERT models for machine translation and natural language understanding. CoRR, abs/1911.06156 URL http://arxiv.org/abs/1911.06156, arXiv:1911.06156.

Tran, H.K.V., Unterkalmsteiner, M., Börstler, J., bin Ali, N., 2021. Assessing test artifact quality—A tertiary study. Inf. Softw. Technol. 139, 106620. http://dx.doi.org/10.1016/j.infsof.2021.106620, URL https://www.sciencedirect.com/science/article/pii/S0950584921000938.

Verma, R.P., Beg, M.R., 2013. 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. http://dx.doi.org/10.1109/ICETET.2013.45.

Wang, C., Pastore, F., Goknil, A., Briand, L., 2020. Automatic generation of acceptance test cases from use case specifications: an NLP-based approach. IEEE Trans. Softw. Eng. 1.

Whalen, M.W., Rajan, A., Heimdahl, M.P., Miller, S.P., 2006. Coverage metrics for requirements-based testing. In: Proceedings of the 2006 International Symposium on Software Testing and Analysis. ISSTA '06, Association for Computing Machinery, New York, NY, USA, pp. 25–36. http://dx.doi.org/10.1145/1146238.1146242.

Xu, J., Zuo, W., Liang, S., Zuo, X., 2020. A review of dataset and labeling methods for causality extraction. In: Proceedings of the 28th International Conference on Computational Linguistics. International Committee on Computational Linguistics, Barcelona, Spain, pp. 1519–1531. http://dx.doi.org/10.18653/v1/2020.coling-main.133, URL https://aclanthology.org/2020.coling-main.133, (Online).

Yang, J., Han, S.C., Poon, J., 2021. A survey on extraction of causal relations from natural language text. CoRR, abs/2101.06426, URL https://arxiv.org/abs/2101.06426.

Zhang, M., Yue, T., Ali, S., Zhang, H., Wu, J., 2014. A systematic approach to automatically derive test cases from use cases specified in restricted natural languages. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (Eds.), System Analysis and Modeling: Models and Reusability. Springer International Publishing, Cham, pp. 142–157.

Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.-V., Batista-Navarro, R.T., 2021. Natural language processing for requirements engineering: A systematic mapping study. ACM Comput. Surv. 54 (3), http://dx.doi.org/10.1145/3444689.

Jannik Fischbach is a Ph.D. student at the Institute of Computer Science of the University of Cologne. From 2019 to 2022, Jannik also worked as a consultant at Qualicen GmbH — a spin-off founded out of the Technical University of Munich focusing on software and systems engineering. In June 2022, he joined Netlight as a consultant. His main research interests include requirements engineering and, in particular, the application of natural language processing methods on requirements artifacts. Jannik holds a Master's degree in Information Systems from the Technical University of Munich.

Julian Frattini is a Ph.D. student at the Blekinge Institute of Technology located in Karlskrona, Sweden. Since 2020 he is working under the supervision of Daniel Mendez in the area of requirements quality, specifically investigating the notion of good-enough requirements engineering. Through the collaboration with Ericsson Karlskrona the research is applied and grounded in practice. Julian holds a Master's degree in Informatics from the Technical University of Munich.

Andreas Vogelsang is full professor for Software and Systems Engineering at the University of Cologne. He received a PhD from the Technical University of Munich. His research interests comprise requirements engineering, model-based systems engineering, and software architectures for embedded systems. He has published over 70 papers in international journals and conferences such as TSE, JSS, IEEE Software, and ICSE. In 2018, he was appointed as Junior-Fellow of the German Society for Informatics (GI). Further information can be obtained from https://cs.uni-koeln.de/sse.

Daniel Mendez is full professor at the Blekinge Institute of Technology, Sweden, and Lead Researcher heading the research division Requirements Engineering at fortiss, the research and transfer institute of the Free State of Bavaria for software-intensive systems and services. After studying Computer Science and Cognitive Neuroscience at the Ludwig Maximilian University of Munich, he pursued his doctoral and his habilitation degrees at the Technical University of Munich. His research is since then on Empirical Software Engineering with a particular focus on interdisciplinary, qualitative research in Requirements Engineering and its quality improvement — all in close collaboration with the relevant industries. He is further editorial board member for EMSE and JSS where he co-chairs the special tracks Reproducibility & Open Science (EMSE) and In Practice (JSS) respectively. Finally, he is a member of the ACM, the German association of university professors and lecturers, the German Informatics Society, and ISERN. Further information is available at http://www.mendezfe.org.

Michael Unterkalmsteiner is a senior lecturer at the Blekinge Institute of Technology, Sweden, where he also received a PhD in Software Engineering. He has been researching Software Engineering since 2009, focusing in particular on the coordination between requirements engineering and software testing. His research work is shaped by empirical problem identification, in-depth analysis of the state-of-art and practice, and collaborative solution development. This empirical, practice-driven approach has led to innovative and scalable solutions. His current research focuses on designing and implementing automated decision support systems for software engineers. Further information is available at https://lmsteiner.com.

Andreas Wehrle is a software engineer at Allianz Deutschland AG. His main interest is the application of natural language processing methods to requirements. Andreas holds a Master's degree in Information Systems from the Technical University of Munich.

Pablo Restrepo Henao is an IT consultant for software engineering and machine learning at Netlight Consulting. He has worked as software engineer and technical lead in multiple companies and is currently pursuing his Master's degree in Computer Science at the Technical University of Munich. His main research interest is the application of natural language processing techniques in the software engineering area.

Parisa Yousefi is a line manager and owner of architecture with Business Solution System (BSS) in Ericsson. Having a background as a developer, her interest are with AI/ML, platform related advances and new technologies as well as core software engineering principles and methodologies such as Agile, Test Driven development & requirement engineering.

Tedi Juricic is a technical quality assurance officer in functional testing within the Business Support Solution Charging and Billing unit at Ericsson. He is responsible for asserting the official quality stamp on the work packages as well as assuring the overall quality of the BSS product.

Jeannette Radduenz is a quality and test manager within the Platform Management & Testing Services unit at Allianz Technology. She is mainly responsible for planning, coordination, and control of test activities.

Carsten Wiecher is a development engineer at KOSTAL Automobil Elektrik GmbH & Co. KG. Since 2013, Carsten has been working in different development projects in the field of e-mobility. His main focus is in the area of software integration for complex electronic control units. Since 2018, Carsten is also a research associate at the IDiAL institute which is part of Dortmund University of Applied Sciences and Arts. Since 2020, he is involved in a research project with KOSTAL focusing on model-based systems engineering, requirements analysis and test specification. Carsten holds a Master's degree in Information Technology from Dortmund University of Applied Sciences and Arts.