applied sciences

MDPI

Article

Automated Conditional Statements Checking for Complete Natural Language Requirements Specification

Chun Liu [1,2], Zhengyi Zhao [1], Lei Zhang [3] and Zheng Li [1,*]

[1] School of Computer and Information Engineering, Henan University, Kaifeng 475000, China; liuchun@henu.edu.cn (C.L.); zzy@henu.edu.cn (Z.Z.)
[2] Henan Industrial Technology Academy of Spatio-Temporal Big Data, Henan University, Zhengzhou 450046, China
[3] Institute of Spacecraft System Engineering, Beijing 100094, China; xxzhangleixx@126.com
* Correspondence: lizheng@henu.edu.cn

Abstract: Defects such as the duality and the incompleteness in natural language software requirements specification have a significant impact on the success of software projects. By now, many approaches have been proposed to assist requirements analysts to identify these defects. Different from these approaches, this paper focuses on the requirements incompleteness implied by the conditional statements, and proposes a sentence embedding- and antonym-based approach for detecting the requirements incompleteness. The basic idea is that when one condition is stated, its opposite condition should also be there. Otherwise, the requirements specification is incomplete. Based on the state-of-the-art machine learning and natural language processing techniques, the proposed approach first extracts the conditional sentences from the requirements specification, and elicits the conditional statements which contain one or more conditional expressions. Then, the conditional statements are clustered using the sentence embedding technique. The conditional statements in each cluster are further analyzed to detect the potential incompleteness by using negative particles and antonyms. A benchmark dataset from an aerospace requirements specification has been used to validate the proposed approach. The experimental results have shown that the recall of the proposed approach reaches 68.75%, and the F1-measure (F1) 52.38%.

Keywords: software requirements specification; requirements quality; requirements analysis; natural language; incompleteness detection

Citation: Liu, C.; Zhao, Z.; Zhang, Z.; Li, Z. Automated Conditional Statements Checking for Complete Natural Language Requirements Specification. *Appl. Sci.* **2021**, *11*, 7892. https://doi.org/10.3390/app11177892

Academic Editor: Alberto Rodrigues Da Silva

Received: 27 July 2021
Accepted: 21 August 2021
Published: 26 August 2021

1. Introduction

It has been shown that the quality of software requirements specification has a significant impact on the success of software projects [1]. The duality, repetition, and incompleteness in a software requirements specification (SRS, or just "requirements specification") which is often written in natural language may lead to the failure of projects or affect the dependability of the software systems developed.

To ensure the quality of software requirements specification, researchers have proposed non-natural language approaches to specify software requirements, for example, the formal languages of Z [2] and Petri Nets [3], the graphical languages of UML [4,5] and SysML [6], the scenario-based [7] and the table-based [8] approaches. However, it is not easy to apply these non-natural language approaches in practice. On the one hand, requirements analysts and stakeholders need to familiarize themselves with these approaches beforehand. On the other hand, since natural language is what the requirements analysts and the stakeholders use when thinking and communicating, applying these non-natural language approaches requires analysts and stakeholders to switch between these non-natural languages and the natural language. This makes it more difficult and inconvenient to apply these approaches, and leads to the result that most software requirements specifications are still written in natural language in practice. The investigation of

Luisa et al. [9] has shown that 95% of software requirements in industry are described in natural language or controlled natural language [10,11]. In this case, it is of great significance for the methods which can assist requirements analysts to identify the defects in the natural language requirements.

By now, many approaches have been proposed for this purpose by different researchers. For example, Chantree et al. [12] proposed methods to discover the presence of duality in natural language requirements; Gervasi et al. [13] proposed the method which transforms the requirements into propositional logic to discover potential conflicts in natural language requirements; Arora et al. [14,15] presented a method to detect whether the requirements described in natural language conform to the qualified templates with the natural language processing techniques; Misra et al. [16] proposed a series of methods to detect terminological inconsistencies in the natural language requirements. Due to the arbitrariness that requirements analysts use the natural language to describe software requirements, existing methods tend to detect some specific types of requirement defects.

In this paper, we focus on detecting the requirements incompleteness implied by the conditional statements. In domains such as aerospace, many software requirements are often expressed in terms of conditional statements. For example, one requirement expressed in terms of a conditional statement is: "When there are GPS events to report, the communication system shall send a detailed report with event parameters to the ground station." For this requirement, the analysts need to specify at least one requirement about what actions to be adopted when there are no GPS events to report. Otherwise, the requirements specification is incomplete. This could lead to the result that the ground station cannot judge the status of the communication system and the aircraft.

This paper proposes a sentence embedding- and antonyms-based approach for detecting the requirements incompleteness implied by the conditional statements. The basic idea is that when one condition is stated, its opposite condition should also be there. Otherwise, the requirements specification is incomplete. Based on state-of-the-art machine learning and natural language processing techniques, the proposed approach detects the potential requirements incompleteness by clustering the conditional statements and finding the opposite of the words in the clusters. Firstly, the conditional sentences are extracted from the natural language requirements specification. The sentences are parsed to elicit conditional statements containing one or more conditional expressions. Then, the conditional statements are clustered based on the sentence embedding technique. Finally, the conditional statements in each cluster are analyzed to detect the potential incompleteness by using the negative particles and the antonyms. A benchmark dataset from an aerospace requirements specification has been used to validate the proposed approach. The experimental results have shown its effectiveness in terms of precision, recall and F1.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to the related works. Section 3 describes the proposed approach in detail. Section 4 presents the evaluation of the proposed method and provides a discussion of the experimental results. Finally, Section 5 summarizes the paper and the future work.

2. Related Work

There are already many works that present different methods to detect defects in the natural language requirements specification. In this section, we will give a brief introduction to these works.

To find missing and conflicting requirements, Moitra et al. [17] described a tool ASSERT™ which was developed by General Electric Company. This tool requires the analysts to define the entities and variables involved in the requirements description using Semantic Application Design Language (SADL) and then automatically validate them using the ACL2 formalism. Filipovikj et al. [18] proposed the SMT formal approach to detect requirement conflicts. This approach relies on a template-based tool [19] to convert requirements described in natural language into a form acceptable to the formal approach,

and the conversion process requires human intervention to determine how to format the requirements based on the templates.

Gervasi and Zowghi et al. [13] converted requirements described in natural language into different propositions by defining a set of rules for each of the conditions and actions in the requirement descriptions based on propositional logic and then reasoned to identify potentially conflicting requirements. Kim et al. [20] also have proposed a method for requirements conflict detection. This method consists of two main steps. The first step is to discover potential syntactically conflicting requirements by defining some rules. The second step is to analyze whether there is a conflict through some heuristic questions.

Moser et al. [21] used ontology-based reasoning to discover requirement conflicts. It requires the prior establishment of an ontology to discover some common words and the relationships between words, and uses the relationship in the ontology to reason the conflicts between the requirements. To find potential requirements conflicts between stakeholders and subsystems, an analytical framework was introduced by Viana et al. [22]. This analytical framework focuses on discovering conflicts in resource utilization by using resources as a link. It requires to first define a resource ontology, then discovers overlapping requirements, and finally detects conflicts between requirements. However, this analysis framework mainly detects dynamic conflicts during the operation of the software. Arora et al. [14,15] proposed an approach that is based on the part-of-speech of each phrase in the requirements, to check whether the requirement descriptions follow the Rupp template [23] and the EARS template [24,25]. The method's key is to determine whether the requirement sentences have "noun + verb" collocation. If the "noun + verb" collocation is present, the sentence is considered reasonable. It then goes on to judge the other parts, including conditionals, etc. The authors mentioned that their method cannot detect semantic inconsistencies.

Misra et al. [16] focused on the problem of inconsistency of phrases in software requirements described in natural language. They proposed different findings and solutions for different cases, such as the inconsistency between phrases and abbreviations of phrases and semantically similar phrases to describe the same object. Hui Yang et al. [26] focused on metaphorical ambiguity. A heuristic approach is used to capture information about a particular text interpretation. Then, a classifier is constructed to identify sentences that contain ambiguities, and the classifier is used to alert requirement writers to texts that carry the risk of misinterpretation. Chantree et al. [12] created a dataset of ambiguous phrases from a requirements corpus and used heuristics to alert authors to the potential dangerous ambiguities. Ferrari et al. [27] proposed an approach for detecting pragmatic ambiguity in requirements specification. The method extracts different knowledge graphs to analyze each requirement sentence looking for potential ambiguities.

Besides these various methods, a number of software tools have been also developed for detecting the defects from natural language requirements specification according to the requirements specification guidelines such as EARS and INCOSE [28–31]. For example, RAT [32] implements an intellisense way of writing that can detect inconsistencies, ambiguity, and duplicates in requirements documents. Through detection, enumeration and classification of all units of measure and noun phrases, QVscribe [33] verifies their correct use and location in the requirements. It uses the EARS method to ensure that the natural language requirements in the software requirements specification are simple and complete. IBM RQA [34] uses the Watson natural language processing technology to deal with escape clauses, missing units, missing tolerances, ambiguity and other issues in requirements documents.

Unlike those detection methods mentioned above, we focus on the incompleteness implied by the conditional statements. The proposed approach uses clustering methods to cluster the conditional statements and find similar conditions, and then detects incompleteness by identifying the opposite of the negative particles [35] or the words in an antonyms lexicon.

3. The Approach

In this section, we present the details of the approach. Figure 1 presents the framework of our proposed approach. There are three phases in the proposed approach: conditional statements extraction, conditional statements clustering, requirements incompleteness detection.

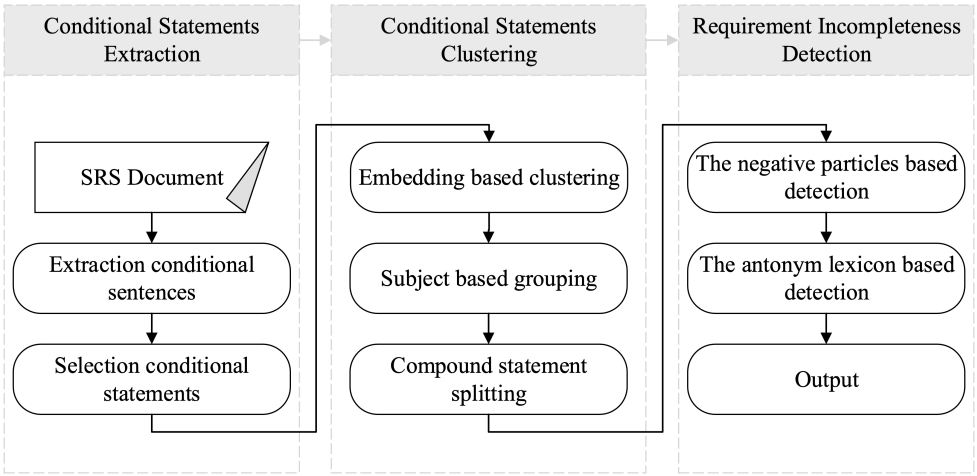

Figure 1. The framework of the proposed approach.

The phase of *conditional statements extraction* takes the natural language requirements specification as input. Its purpose is to extract the conditional sentences and divide the sentences into conditional statements and action statements. The conditional statements describe one or more conditional expressions which describe the occurrence of some events, and the action statements describe the actions that should be adopted when the conditional statements are true. The phase of *conditional statements clustering* takes the conditional statements as input. It obtains the distributed representations of the conditional statements through embedding techniques such as Skip-Thought [36], clusters the semantically related statements, and finds the statements describing the same events. The phase of *requirements incompleteness detection* detects the potential incompleteness in each group of conditional statements based on the negative particles and the antonyms. We detail each phase as follows.

3.1. Conditional Statements Extraction

This phase is to extract the conditional sentences and obtain the conditional statements of the sentences. In English, "condition" means that something else (actions) can happen after one thing happens (event). Therefore, the conditional sentences often contain three parts: the conditional statement, the action statement, and the subordinating conjunction, as shown in Figure 2. The subordinating conjunctions are often the indicators of conditional sentences. The common subordinating conjunctions include "when", "if", "while", and "where". They are often used before the conditional statements and indicate that the following statement is a conditional statement describing the occurrence of some events. They can be located at the beginning of the sentences, but also can be after the action statements in practice.

Taking the requirements specification document written in natural language as input, the proposed approach first extracts the conditional sentences according to the four subordinating conjunctions of "when", "if", "while", and "where". Once the conditional sentences are extracted, the conditional statements are obtained by dividing the sentences into conditional statements and action statements. After that, all the conditional statements are further lowercased with the NLTK [37] tool.

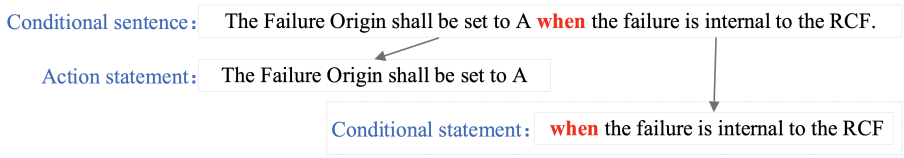

Figure 2. An example of a conditional sentence segmentation.

3.2. Conditional Statements Clustering

This phase clusters the conditional statements extracted. The purpose is to group the conditional statements which refer to the same subjects. This paves the way for detecting the requirements incompleteness implied by the conditions.

3.2.1. The Embedding-Based Clustering

To cluster the conditional statements, while keeping the original versions of these conditional statements, we first process them with the following steps provided by NLTK tool:

- Tokenization: break down statements into words;
- Part of Speech (POS) tagging: label words with known lexical categories;
- Words selection: keep only the verbs, nouns, adjectives;
- Stop words removal: remove commonly used words;
- Lemmatization: make the words to general form.

After these preprocessing steps, the distributed representations of each statement are obtained through the sentence embedding techniques such as Skip-Thought. Then, the statements are clustered based on the clustering methods in machine learning. This clustering process contributes to finding similar conditional statements quickly.

3.2.2. The Subject-Based Grouping

In order to further find these conditional statements which refer to the same subjects, we group the conditional statements in one cluster according to whether they have the same noun words. As shown in Figure 3, these noun words are often the subjects of the conditional statements [38]. In this way, the statements in one cluster could be divided into different groups.

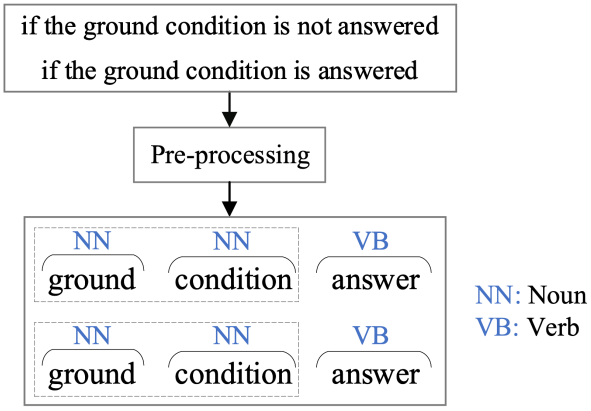

Figure 3. The part-of-speech tagging of the conditional statements.

3.2.3. Compound Statement Splitting

For conditional statements (the original versions) in one group, they may be the compound sentences which consist of several independent conditional clauses. To address the compound statements, we split them to make that all conditional statements in one group are simple sentences.

There are two classes of compound statements according to whether they share the same subjects.

- *Shared subjects:* It is composed of two or more independent conditional clauses connected by coordinating conjunctions. However, for the second and subsequent conditional clauses, the subjects are omitted;
- *Non-shared subjects:* It comprises two or more independent conditional clauses connected by coordinating conjunctions. Each independent conditional clause has its own subject.

Different methods are adopted to split the compound statements. We split the statements by the coordinating conjunctions directly for the statements that do not share the same subjects. At the same time, for the statements that share the same subjects, we not only split the statements by the coordinating conjunctions, but also take the subject of the first clause as the subject of the second and subsequent clauses.

3.3. Requirements Incompleteness Detection

Based on the grouped conditional statements, this phase detects the potential incompleteness implied by the conditional statements. The basic idea is that if one condition is stated, its opposite condition should also be stated. Thus, for example, if one positive statement is described, the related negative statement should also be given and vice versa. To achieve this goal, the potential incompleteness is detected by using the negative particles and the antonyms, respectively. The detection process is shown in Figure 4. We detail the process as follows.

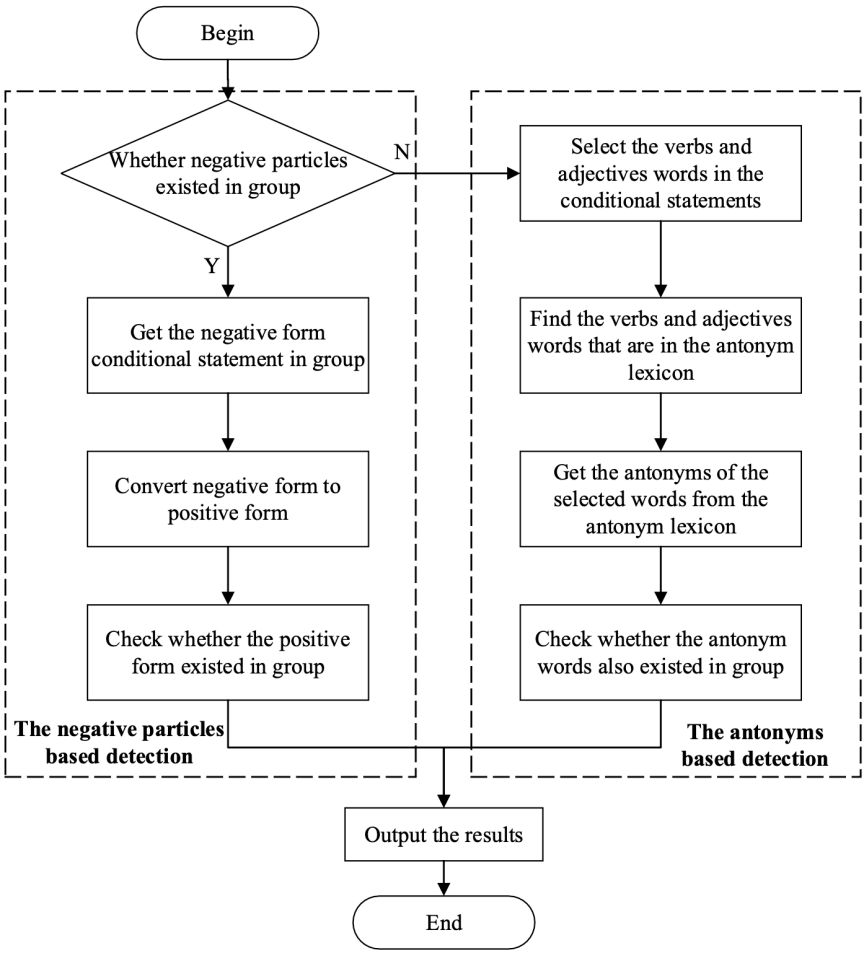

Figure 4. The process of the requirements incompleteness detection.

3.3.1. The Negative Particles-Based Detection

The negative particles include "no", "not", "do not", "does not", "did not", "don't", "doesn't", "didn't". When the conditional statements contain these negative particles, they

are expressing the negative meanings. In this case, for the completeness of the requirements, the corresponding conditional statements without the negative particles should be also there. Otherwise, the requirements are not complete.

Therefore, for each statement (the original version of the conditional statements which are not preprocessed) in one group, we check whether they contain the negative particles. If the negative particles are found, we further check whether the corresponding statements without the negative particles are there. If the corresponding statements are found, the statements and their corresponding statements are considered as the complementary statements and the requirements implied by them are complete. Otherwise, the requirements implied by the statements are considered as potential incomplete requirements.

3.3.2. The Antonyms-Based Detection

Besides the negative particles, an antonym lexicon is constructed and utilized for incompleteness detection. We have built the antonym lexicon according to "The Merriam-Webster Dictionary of Synonyms and Antonyms" [39]. A total of 326 words related to engineering were extracted to create the antonym lexicon, and some of the antonyms are shown in Table 1. In addition, to prevent potential errors caused by the high-frequency words with multiple meanings such as "first" and "last", some of the high-frequency antonyms are eliminated.

Table 1. The antonyms in the antonym lexicon.

hide-appear	build-destroy	complicated-regulation
abide-violate	enhance-alleviates	approval-prohibition
nadir-zenith	ascends-down	mandatory-optional
forbid-permit	mutual-separate	mobile-immobile
permanent-temporary	contrary-similar	interrupt-continue
lateral-vertical	decode-encode	incongruity-compatibility
disappear-appear	dismantle-construction	direct-indirect
activate-failure	continue-interrupt	relieve-enhanced
accurate-inaccurate	divestiture-restore	agree-disagree
valid-invalid	erasing-preserve	disadvantage-advantage
accept-decline	huddle-disperse	invalidation-efficacious
opacity-transparent	fall-rise	acquire-bereavement
approve-disapprove	legitimate-illegitimate	propagate-restrain
air-ground	hinder-unobstructed	progress-regress
fasten-unfasten	impede-expedite	fix-replace
abate-aggravation	excess-lack	indirect-direct
equal-unequal	partial-entire	caution-indiscretion
forward-backward	exit-entrance	retract-confirm
internal-external	track-lose	inadequacy-abundance
deliver-collect	lock-unlock	sporadic-frequently
disobey-obey	empty-fill	horizontal-vertical
gargantuan-negligible	discover-miss	...

With the antonym lexicon, the proposed approach first checks whether each statement in one group contains the negative particles. If there are no negative particles, it then tokenizes the statement, removes the stop-words, tags the part of speech of the words, keeps only the verbs and adjectives words, and makes lemmatization of the words. This is because the antonym words are often the verbs and adjectives words. Subsequently, for each word that is left, if it is in the antonym lexicon, we will find its antonym in other statements in the same group. If its antonym is found, the requirement implied by the statement is considered complete; otherwise, it is not.

4. Empirical Evaluation

We evaluate our approach with a real-life case study. In this section, we present the research questions, the study design, the results and analysis.

4.1. Research Questions(RQs)

Our evaluations aim to answer the following two questions:

RQ1: What is the optimal solution for each machine learning module?

Two machine learning modules are used in the proposed approach: sentence embedding and statements clustering. For each module, we need to choose from several alternative implementations. The purpose of RQ1 is to determine which implementation will produce better results for each module.

RQ2: How does the performance of our approach perform?

When the implementations are selected for these machine learning modules, the purpose of RQ2 is to observe the performance of our approach in checking the requirements incompleteness implied by the conditions.

4.2. Study Design

4.2.1. The Datasets

To answer the research questions, we take a standard requirements statement document about a Spacecraft as an example dataset. This document consists of thousands of requirements. After extracting the requirements according to the keywords of "when", "if", "while", and "where", a total of 130 conditional sentences were selected.

For this dataset of conditional sentences, two members of our research group have been called to cluster these conditional statements according to their subjects manually. Accordingly, these conditional statements have been divided into 75 groups. During this process, incompleteness has been found in eight groups. Moreover, to increase the number of cases of incompleteness, we have also randomly selected eight conditional sentences and deleted them from the groups where they were located.

4.2.2. The Implementation

To implement the proposed approach, we used the deep learning framework of Anaconda [40] and wrote the program using Python. The Scikit-learn [41] library is used to provide the implementations of the clustering algorithm; the Gensim [42] library is used to provide the implementations of the sentence embedding approaches. Table 2 shows the versions of the tools we used.

Table 2. The versions of the tools.

Tool	Versions
Anaconda	4.9.2
Python	3.8.5
Scikit-learn	0.23.2
Gensim	3.8.3
NLTK	3.5

4.2.3. The Metrics

To evaluate the effectiveness of the proposed approach, the metrics of precision, recall and $F1$ are used. *They are defined as follows*:

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

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3)$$

In the definitions, TP means the number of positive samples correctly classified, FP is the number of negative samples incorrectly labeled as positive samples, and FN is the number of positive samples incorrectly labeled as negative samples. This means that *precision* measures how many samples are correctly classified among these samples predicted as positive samples, *recall* measures how many positive samples are correctly classified, and $F1$ combines the values of *precision* and *recall*.

4.3. Descriptions of the Studies

To answer the research questions, several studies have been performed. The details of each study are described below.

4.3.1. Study I: Selection of Sentence Embedding Methods

The proposed approach clusters the conditional statements to group the similar statements. For this purpose, it utilizes the sentence embedding methods to obtain the distributed representations of each statement. There are several candidates for the sentence embedding methods. This study is to compare the performance of these candidates and select the better one for the conditional statements clustering and answer the research question RQ1. We observe the performance of these methods through clustering the conditional statements with the K-Means [43] clustering algorithm. The selected candidates for comparison are as follows:

1. Word2vec+TF-IDF: Word2Vec [44] maps each word in the dataset into a vector, while TF-IDF [45] is a method which can generate a score for each word. In this case, the weighted representations of the conditional statements can be generated by combining the approaches of Word2Vec and TF-IDF. The implementations of Word2Vec and TF-IDF in Gensim library are used. The Word2Vec is a pre-trained model from Google [46].
2. Doc2Vec: Doc2Vec [47] is an unsupervised algorithm that can learn a fixed-length feature representation from sentences. The implementation of Doc2Vec in Gensim library is used.
3. Bert: Bert [48] is an unsupervised algorithm, and a deeply bidirectional system for pre-training NLP. The implementation of Bert in Anaconda is used.
4. Skip-Thought: Skip-Thought is an unsupervised algorithm which provides a general distributed sentence encoder. We downloaded the source code of Skip-Thought from GitHub [49], and used the encoder of the pre-trained model to generate the representations of conditional statements. When using skip-thought, the required python version is 2.7.

To evaluate the performance of above sentence embedding methods, we take the measures of rand index (*RI*) [50] and mutual information (*MI*) which are often used for the evaluation of the clustering algorithms. They are defined as follows:

$$RI = \frac{2(a+b)}{n(n-1)} \quad (4)$$

$$MI(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} log \frac{N|U_i \cap V_j|}{|U_i||V_j|} \quad (5)$$

For rand index, the number of a in Equation (4) is the number of pairs of elements that belong to the same cluster in both the predicted results and the manually grouped results. In the meanwhile, the number of b is the number of pairs of elements that do not belong to a cluster in both the predicted results and the manually grouped results. The number of n is the number of elements to be clustered. The mutual information measures the mutual dependence between two sets. For the definition of Equation (5), U_i is the i-th

cluster generated by the clustering algorithm, and V_j is the j-th cluster of the manually grouped results. The values of both measures are in the range of [0, 1]. The larger, the better.

4.3.2. Study II: Selection of Clustering Methods

There are also many candidate methods for the conditional statements clustering. In this study, we will compare several clustering methods with the same sentence embedding method and observe their performance. This study is to answer the research question RQ1 by selecting one suitable method for the conditional statements clustering. The selected clustering methods used for the comparison are as follows.

1. K-Means: K-Means is the commonly used clustering algorithm based on Euclidean distance. We use the K-Means in the Scikit-learn. The number of clusters is set to 12 according to the sum of the squared errors (SSE).
2. Spectral Clustering: spectral clustering [51] is based on graph theory. We use the implementation of spectral clustering provided by Scikit-learn. The number of clusters is also set to 12.
3. Agglomerative Clustering: agglomerative clustering [52] is a hierarchical clustering approach. We use the implementation of agglomerative clustering provided by Scikit-learn. The number of clusters is also set to 12.
4. DBSCAN: DBSCAN [53] is a density-based clustering approach. The number of clusters is not required for clustering. We use the implementation of DBSCAN provided by Scikit-learn.
5. Affinity Propagation: affinity propagation [54] is a graph-based clustering method, which does not require specifying the number of clusters for clustering. We use the implementation of affinity propagation provided by Scikit-learn.

To evaluate the performance of the clustering algorithms, the metrics of rand index and mutual information are also used.

4.3.3. Study III: The Performance of the Proposed Approach

Based on the results of the above studies, the suitable methods for the conditional statements embedding and clustering will be selected. Then, this study is to evaluate the performance of our approach and answer the research question RQ2 based on the dataset used.

4.4. Results and Analysis

In this section, we describe the results of our studies and answers to the RQs.

4.4.1. RQ1: What Is the Optimal Solution for Each Machine Learning Module?

Table 3 shows the comparison results of the selected sentence embedding methods. The best results are indicated in bold. By comparison, we can find that the Skip-Thought method performs better than the other methods in all metrics. For example, when compared with Word2vec+TF-IDF, the rand index of Skip-Thought is improved by 84.06%, the mutual information by 18.42%. According to this result, the Skip-Thought method is used in the proposed approach for the conditional statements embedding.

Table 3. The results of the comparison between the sentence embedding methods.

	Rand Index	**Mutual Information**
Word2vec+TF-IDF	0.0693	0.6185
Doc2vec	0.0704	0.6382
Bert	0.1197	0.6944
Skip-Thought	**0.1265**	**0.7334**

Table 4 shows the comparison results of the selected clustering methods. The best results are indicated in bold. It can be seen that affinity propagation has the best performance in two metrics. In terms of the rand index, it is 155.72% higher compared to agglomerative clustering. In terms of the mutual information, it is 27.19% higher compared to spectral clustering. This result indicates that it is better to select the affinity propagation method for the conditional statements clustering.

Table 4. The comparison results between the clustering methods.

	Rand Index	**Mutual Information**
K-Means	0.1265	0.7334
Spectral Clustering	0.0658	0.6814
Agglomerative Clustering	0.1215	0.7331
DBSCAN	0.1979	0.8519
Affinity Propagation	**0.3107**	**0.8667**

4.4.2. RQ2: How Does the Performance of Our Approach Perform?

We have selected the Skip-Thought method for the conditional statements embedding and the affinity propagation method for the conditional statements clustering. Based on the dataset used for the evaluation, the performance of the proposed approach is shown in Table 5. From Table 5, it can be seen that the proposed approach reaches 42.31% in precision and 68.75% in recall.

Table 5. The performance of the proposed approach described in Section 3.

	Precision	**Recall**	**F1**
Our approach	0.4231	0.6875	0.5238

From the results shown in Table 5, it can be seen that the proposed approach has achieved a low precision. Compared with the precision, the recall is higher. This is partly because that in most classification problems, high recall rates come at the cost of the decrease of precision [55]. By analyzing the results of the proposed approach, we found that there are several factors that affect the performance of the proposed approach.

1. *Template non-conformance*: The proposed approach finds the opposite conditions by using the negative particles and the antonyms. This means that when the conditional statements have not used the negative particles and antonyms, the opposite conditions will not be found. For example, for the conditional statement "*When there are GPS events to report*", it is difficult for our approach to find its opposite "*When there are none GPS events*".
2. *Unknown words*: We found that there are a lot of proper nouns used in the requirements specification. In most cases, they appear in the form of abbreviations. For a part of the speech tagging of these words, low accuracy is often achieved. This will affect the grouping of the conditional statements.
3. *Incorrect keywords*: When specifying requirements, the analysts may use some words to describe some objects such as a button. For example, for the conditional statement "*When the screen prompts 'DORMANCY LIGHT: ON'*", the word of DORMANCY here refers to a light and not a state. In this case, it is incorrect to detect the incompleteness by finding the antonym of this word. However, our approach cannot identify these cases and this results in some wrongly reported detections.
4. *Ill-formed sentences*: After clustering the conditional statements, the proposed approach groups the conditional statements according to the subjects of the statements. However, in practice, the subjects may be missing and the statements are not complete. For example, for the conditional statement "*When transmitted*", it is obvious that the subject is missing. This will lower the precision of the proposed approach.

Among these factors, the cases of *template non-conformance* and *unknown words* collectively account for approximately 90 percent of the values of FN and FP. For these factors, more improvements should be adopted to address them in the future.

5. Conclusions

When specifying the requirements, there may be many requirements described in terms of conditions. If one condition is stated and its opposite condition is not there, the requirements implied by the condition are incomplete. Focusing on this kind of requirement incompleteness, this paper has proposed a sentence embedding and antonym-based approach for detecting the requirement incompleteness implied by the conditional statements. To detect the incompleteness, the proposed approach first extracts the conditional sentences from the requirements specification, and elicits the conditional statements which contain one or more conditional expressions. Then, the conditional statements are clustered with sentence embedding techniques. The conditional statements in each cluster are further analyzed to detect the potential incompleteness by using the negative particles and the antonyms. A benchmark dataset from an aerospace requirements specification has been used to validate the proposed approach. The validation results have shown that the recall of the proposed approach reaches 68.75%, and the F1 52.38%.

It should be noted that the proposed approach cannot detect all potential incompleteness implied by the conditional statements. For example, the proposed approach finds the opposite conditions by using the negative particles and the antonyms. This leads to the result that, when the conditional statements have not used the negative particles and antonyms, the opposite conditions will not be found. The abbreviations of the proper nouns and the ill-formed sentences in the requirement specifications may also affect the performance of the proposed approach. In the future, more efforts will be adopted to address these factors and improve the proposed approach.

Author Contributions: Conceptualization, C.L. and Z.L.; methodology, C.L.; software, Z.Z.; validation, C.L., L.Z. and Z.Z.; formal analysis, C.L. and Z.Z.; investigation, L.Z.; resources, C.L.; data curation, L.Z.; writing—original draft preparation, C.L.; writing—review and editing, Z.L.; supervision, C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Brooks. No Silver Bullet Essence and Accidents of Software Engineering. *Computer* **1987**, *20*, 10–19. [CrossRef]
2. Davies, J.; Woodcock, J. *Using Z: Specification, Refinement, and Proof*; Prentice Hall International: Hoboken, NJ, USA,1996
3. Peterson, J.L. Petri Nets. *ACM Comput. Surv.* **1977**, *9*, 223–252. [CrossRef]
4. Group, O.M. UML Resource Page. 2021. Available online: http://www.uml.org/ (accessed on 26 March 2021).
5. Holt, J. *UML for Systems Engineering: Watching the Wheels*; IET: London, UK, 2004; Volume 4.
6. Group, O.M. Official OMG SysML Site. 2021. Available online: http://www.omgsysml.org/ (accessed on 27 March 2021).
7. Alexander, I.F.; Maiden, N. *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, 1st ed.; Wiley Publishing: Hoboken, NJ, USA, 2004.
8. Alexander, I.F.; Beus-Dukic, L. *Discovering Requirements: How to Specify Products and Services*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
9. Luisa, M.; Mariangela, F.; Pierluigi, N.I. Market Research for Requirements Analysis Using Linguistic Tools. *Requir. Eng.* **2004**, *9*, 40–56. [CrossRef]
10. da Silva, A.R.; Savić, D. Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Data Entities. *Appl. Sci.* **2021**, *11*, 4119. [CrossRef]

11. Schwitter, R. Controlled Natural Languages for Knowledge Representation. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Beijing, China, 23–27 August 2021; Association for Computational Linguistics: Stroudsburg, PA, USA, 2010; pp. 1113–1121.
12. de Roeck, A.; Willis, A.; Nuseibeh, B.; Chantree, F. Identifying Nocuous Ambiguities in Natural Language Requirements. In Proceedings of the 14th IEEE International Requirements Engineering Conference, Minneapolis, MN, USA, 11–15 September 2006; Computer Society: Los Alamitos, CA, USA, 2006; pp. 59–68. [CrossRef]
13. Gervasi, V.; Zowghi, D. Reasoning about Inconsistencies in Natural Language Requirements. *ACM Trans. Softw. Eng. Methodol.* **2005**, *14*, 277–330. [CrossRef]
14. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F. Automated Checking of Conformance to Requirements Templates Using Natural Language Processing. *IEEE Trans. Softw. Eng.* **2015**, *41*, 944–968. [CrossRef]
15. Arora, C.; Sabetzadeh, M.; Briand, L.; Zimmer, F. Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Trans. Softw. Eng.* **2017**, *43*, 918–945. [CrossRef]
16. Misra, J. Terminological inconsistency analysis of natural language requirements. *Inf. Softw. Technol.* **2016**, *74*, 183–193. [CrossRef]
17. Moitra, A.; Siu, K.; Crapo, A.; Chamarthi, H.; Durling, M.; Li, M.; Yu, H.; Manolios, P.; Meiners, M. Towards Development of Complete and Conflict-Free Requirements. In Proceedings of the 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AB, Canada, 20–24 August 2018; pp. 286–296. [CrossRef]
18. Filipovikj, P.; Rodriguez-Navas, G.; Nyberg, M.; Seceleanu, C. SMT-Based Consistency Analysis of Industrial Systems Requirements. In Proceedings of the Symposium on Applied Computing, Maracas, Morocco, 4–6 April 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1272–1279. [CrossRef]
19. Filipovikj, P.; Nyberg, M.; Rodriguez-Navas, G. Reassessing the pattern-based approach for formalizing requirements in the automotive domain. In Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE), Karlskrona, Sweden, 25–29 August 2014; pp. 444–450. [CrossRef]
20. Kim, M.; Park, S.; Sugumaran, V.; Yang, H. Managing requirements conflicts in software product lines: A goal and scenario based approach. *Data Knowl. Eng.* **2007**, *61*, 417–432. [CrossRef]
21. Moser, T.; Winkler, D.; Heindl, M.; Biffl, S. Automating the Detection of Complex Semantic Conflicts between Software Requirements. In Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering, Miami Beach, FL, USA, 7–6 July 2011
22. Viana, T.; Zisman, A.; Bandara, A.K. Identifying Conflicting Requirements in Systems of Systems. In Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference (RE), Lisbon, Portugal, 4–8 September 2017; pp. 436–441. [CrossRef]
23. Pohl, K. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam-Foundation Level-IREB Compliant*; Rocky Nook, Inc.: San Rafael, CA, USA, 2016.
24. Mavin, A.; Wilkinson, P.; Harwood, A.; Novak, M. Easy Approach to Requirements Syntax (EARS). In Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 31 August–4 September 2009; pp. 317–322. [CrossRef]
25. Mavin, A.; Wilkinson, P. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, Sydney, Australia, 27 September–1 October 2010; pp. 277–282. doi:10.1109/RE.2010.39 [CrossRef]
26. Yang, H.; De Roeck, A.; Gervasi, V.; Willis, A.; Nuseibeh, B. Analysing Anaphoric Ambiguity in Natural Language Requirements. *Requir. Eng.* **2011**, *16*, 163. [CrossRef]
27. Ferrari, A.; Gnesi, S. Using collective intelligence to detect pragmatic ambiguities. In Proceedings of the 2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, 24–28 September 2012; pp. 191–200. [CrossRef]
28. Efremov, A.; Gaydamaka, K. Incose guide for writing requirements. Translation experience, adaptation perspectives. In Proceedings of the CEUR Workshop Proceedings, Como, Italy, 9–11 September 2019; pp. 164–178
29. 21 Top Engineering Tips: How to Write an Exceptionally Clear Requirements Document. White Paper. QRA Corp, 2018. Available online: https://www.qracorp.com/write-clear-requirements-document/ (accessed on 17 August 2021).
30. Anderberg, M.R. *Guide for Writing Requirements*; INCOSE Publications Office: San Diego, CA, USA, 2019 ; Volume 3.
31. *Automating the INCOSE Guide for Writing Requirements*; White Paper; QRA Corp: Halifax, NS, Canada, 2019.
32. RAT—AUTHORING Tools. 2021. Available online: https://www.reusecompany.com/rat-authoring-tools (accessed on 16 August 2021).
33. QVscribe. 2021. Available online: https://qracorp.com/qvscribe/ (accessed on 15 August 2021).
34. IBM Engineering Requirements Quality Assistant. 2021. Available online: www.ibm.com/products/requirements-quality-assistant (accessed on 16 August 2021).
35. Thompson, S.A. A Discourse Explanation for the Cross-linguistic Differences in the Grammar of Interrogation and Negation. *Case Typology Gramm. Honor. Barry J. Blake* **1998**, *309*, 341.
36. Kiros, R.; Zhu, Y.; Salakhutdinov, R.; Zemel, R.S.; Torralba, A.; Urtasun, R.; Fidler, S. Skip-Thought Vectors. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–10 December 2015
37. Bird, S.; Klein, E.; Loper, E. *Natural Language Processing with Python*; O'Reilly Media, Inc.: Newton, MA, USA , 2009.
38. Oxford, O.E. *Oxford English Dictionary*; Oxford University Press: Oxford, UK, 2009.
39. Cornog, M.W. *The Merriam-Webster Dictionary of Synonyms and Antonyms*; Merriam-Webster, Inc.: Springfield, MA, USA, 1992

40. Anaconda Individual Edition. 2021. Available online: https://www.anaconda.com (accessed on 19 February 2021).
41. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
42. Řehůřek, R.; Sojka, P. Software Framework for Topic Modelling with Large Corpora. In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, 22 May 2010; ELRA: Valletta, Malta, 2010; pp. 45–50
43. Macqueen, J. Some Methods for Classification and Analysis of Multivariate Observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, 21 June–18 July 1967; Volume 1, pp. 281–297.
44. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781.
45. Rajaraman, A.; Ullman, J.D. *Mining of Massive Datasets*; Cambridge University Press: Cambridge, UK, 2011.
46. Google Code: word2vec. 2013. Available online: https://code.google.com/archive/p/word2vec/ (accessed on 18 February 2021).
47. Le, Q.V.; Mikolov, T. Distributed Representations of Sentences and Documents. *arXiv* **2014**, arXiv:1405.4053.
48. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805.
49. Zhu, Y.; Kiros, R.; Zemel, R.; Salakhutdinov, R.; Urtasun, R.; Torralba, A.; Fidler, S. Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books. *arXiv* **2015**, arXiv:1506.06724.
50. Rand, W.M. Objective Criteria for the Evaluation of Clustering Methods. *J. Am. Stat. Assoc.* **1971**, *66*, 846–850. [CrossRef]
51. von Luxburg, U. A Tutorial on Spectral Clustering. *arXiv* **2007**, arXiv:0711.0189.
52. Anderberg, M.R. *Cluster Analysis for Applications: Probability and Mathematical Statistics: A Series of Monographs and Textbooks*; Academic Press: Cambridge, MA, USA, 2014; Volume 19.
53. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, OR, USA, 2–4 August 1996 ; pp. 226–231.
54. Frey, B.J.; Dueck, D. Clustering by Passing Messages between Data Points. *Science* **2007**, *315*, 972–976. [CrossRef] [PubMed]
55. Buckland, M.; Gey, F. The Relationship between Recall and Precision. *J. Am. Soc. Inf. Sci.* **1994**, *45*, 12–19. [CrossRef]