CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Detecting inconsistencies of safety artifacts with Natural Language Processing

Bachelor of Science Thesis in Software Engineering and Management

XUNI HUANG

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

This research presents how to detect inconsistencies between safety artifacts by using Natural Language Processing.

Supervisor: Jan-Philipp Steghöfer, Peter Ljunglöf
Examiner: Richard Berntsson Svensson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Detecting inconsistencies of safety artifacts with Natural Language Processing

Xuni Huang
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gushuaxu@student.gu.se

***Abstract*—This paper investigates a method that helps detect inconsistencies between safety-critical systems' textual *safety artifacts* that safety cases rely on by involving NLP techniques. A design science research study was conducted in three iterations. I evaluate the method by conducting different experiments. The designed artifact identifies inconsistencies between different texts that are connected by trace links by checking similarities and word vectors of the texts. The results indicate that involving NLP technique word embedding can help with consistency classification. In conclusion, NLP techniques may help detect inconsistencies between *safety artifacts* that safety cases rely on, which is helpful to reduce the risk of system failures.**

***Keywords*—Inconsistencies, Safety-critical systems, Natural language processing, Classification**

I. Introduction

Safety-critical systems are developed according to functional safety standards, which require a safety assurance case. For instance, ISO 26262 is the safety standard in the automotive domain, DO-178B for avionics, and IEC 62304 for medical devices. Modern safety-critical systems are complicated systems built with a lot of different components [1]. As systems in different domains become more and more complicated, the consistency between *safety artifacts* that are related to safety cases may not be easy to maintain [2]. Consistency here means that any changes that happen in the *safety artifact* should be reflected in other *safety artifacts*. In particular, each *safety artifact* should always semantically be coherent and synchronized with other *safety artifacts* such as requirements, use cases, and designs [3], [4].

Natural language processing (NLP), the study that applies various computational methods to understand and produce different human languages, is nowadays widely used in various domains. In [5], the authors mention that some NLP tools are good at checking the consistency of complex data which will decrease the time consumed on maintenance. Therefore, applying different NLP techniques to *safety artifacts* of safety-critical systems that safety cases rely on, is a promising method to detect inconsistencies of safety cases.

In this research, I developed a method that assists the detection of inconsistencies between *safety artifacts* that are connected by traceability links within the MobSTr dataset developed by Maro et al. [6], by involving NLP techniques. When evaluating the method, the results indicate that using NLP techniques can help with identifying the inconsistencies between textual *safety artifacts*. This is very beneficial for safety-critical systems. If inconsistencies between *safety artifacts* can be detected, they can be solved by the engineers. Detecting the inconsistencies is the first step for ensuring consistency in a safety-critical system. Ensuring consistency can mitigate the risk of loss of life and injury. Furthermore, this paper can act as useful guidance on how to achieve inconsistency detection by using NLP technology, as well as a solid foundation for improving consistency assurance in the future.

In this paper, Section II presents the background regarding this research. Section III demonstrates the related work of this research. Section IV is concerned with the methodology used for this research. The results of each iteration are described in Section V. Section VI focuses on a discussion regarding this research. This paper is concluded with a conclusion in Section VII.

II. Background

A. Safety-critical systems & Safety cases

Safety-critical systems are systems whose failures can lead to loss of life, serious property damage, or environmental damage [7]. As stated in [8], "Safety cases are comprised of all related safety engineering artifacts that argue that a system is safe", and the context of the argument is relevant to the context of the system requirements. Therefore, when inconsistencies appear between these *safety artifacts* when they change, the argument of why a system is "safe" may not be correct and complete anymore. A safety-critical system whose safety is not assured can cause injuries and death. Therefore, for safety-critical systems such as automotive vehicles, aerospace, and medical systems, ensuring that the *safety artifacts* that safety cases rely on remain consistent is extremely important.

Requirements are often written in natural language [9], as well as commit messages, code comments, release notes, and other similar textual contents. They might not be structured and clear [10], and can very easily contain mistakes. Because of this, inconsistencies between these *safety artifacts* can easily occur. If inconsistencies are not detected in time, it can harm system safety. Besides, the ambiguous or unstructured textual content can mislead the development team to introduce or ignore some bugs that will place the system at risk.

As arguments to verify that the system is safe, safety cases argue that the system meets the safety requirements and these safety requirements are complete and sufficient. Accordingly, if changes in safety requirements and other textual contents such as system failures are not reflected in safety cases, inconsistencies of safety cases can appear. Safety cases still argue that the system is safe depending on the old safety requirements, while there might be some changes in the new safety requirements. This is a big safety risk.

As stated in [11], safety-critical systems which do not meet safety requirements can lead to disasters. For example, a train accident happened in Singapore since a safety protection software feature was removed accidentally [12]. Therefore, assurance of the consistency between *safety artifacts* that safety cases reply on, and safety requirements is crucial to prevent accidents.

B. Traceability information model of the dataset

This subsection presents the traceability information model (TIM) of the MobSTr dataset [6] for which I designed the artifact. Fig. 1 shows parts of the TIM of this dataset. As shown in Fig. 1, the safety case has traceability links to safety requirement, hazard, and FMEA (Failure Mode and Effects Analysis). Hazard has traceability links to safety goal and effect. Effect is a consequence of a failure of a component within the system, which can lead to hazards. Safety goal, which is derived from hazard, inherits from safety requirement. These links indicate that safety case relies on *safety artifacts* effect, hazard, safety goal, and safety requirement. In order to ensure safety cases remain consistent when there are changes within the system, we have to ensure the consistency between *safety artifacts* that safety cases rely on as well.

Therefore, the designed artifact focuses on detecting inconsistencies between textual *safety artifact* components effect, hazard, safety goal, and safety requirement within Fig. 1. Each effect is connected to a hazard, each hazard is connected to a safety goal, and each safety goal is connected to safety requirements. Since these trace links are transitive, as long as effect and safety requirement remain consistent with each other, all other trace links are consistent as well. Thus, the designed artifact check the consistency between all these *safety artifacts* by checking the consistency between effect and safety requirement.

C. Textual data processing

When detecting inconsistencies between textual *safety artifacts* by involving NLP, some special techniques are needed. This subsection presents the techniques and the terminologies that are related to Natural language processing.

1) Natural language processing (NLP)

In order to investigate how to detect inconsistencies with NLP, first, it is important to understand what NLP is. Natural languages are languages that humans use in daily life for communications [5]. NLP covers different kinds of computer manipulation of natural language [5]. For example, human-machine translation and speech recognition are products that utilize NLP techniques [13].

2) Text similarity

Some advanced NLP techniques can calculate the similarity between texts. As a promising approach to detect inconsistencies between texts, text similarity assesses how similar two texts are. This kind of similarity can be lexical similarity and semantic similarity [14]. Lexical similarity depends on the character sequence of words [14], while semantic similarity depends on the similarity of the meaning of the content. Semantic textual similarity (STS) measures the degree to which two sentences are semantically equivalent to each other [15]. Some methods that are used to calculate text similarity require some preprocessing of the raw texts. Normalizing texts is one type of text preprocessing.

3) Text normalization

Texts can include non-standard words such as numbers, abbreviations, and punctuation, which may increase the ambiguity of the texts [16]. Therefore, a preprocessing of texts to convert texts into standard formats, called text normalization is needed. The first step of text normalization is tokenization, which transfers the raw text data into a list of words and punctuation [5]. After getting this list, we can remove stop words and punctuation to retain the most important ordinary words. Lowercasing the text also belongs to text normalization. By converting the text into lowercase, for example, “No” and “no” will be identified as identical [17].

4) Word embedding

Other than text similarity calculated by NLP, word embedding [18] is also a promising technique of NLP that may be used to detect text inconsistencies. Word embedding maps each word in natural language to a high dimensional real-valued vector. In a sentence, every word corresponds to some vectors, which converts the text analysis to vector comparison.

D. Similarity metrics

For calculating text similarity, there exist different kinds of similarity metrics. In this subsection, I present the similarity metrics that were used in this research.

1) Levenshtein distance

A metric that measures the text distance. Levenshtein distance [19] is calculated by the number of common characters two words have. For example, the Levenshtein distance between “cat” and “hat” is 2, since these two words have two common characters.

2) Jaccard index

A metric that measures similarity between two finite sample sets [20]. Jaccard index [20] is calculated by using the intersection of two sample sets divided by the union of two sample sets [20]. For example, for text1 = “I have a cat” and text2 = “I have a hat”, the Jaccard index will be $\frac{3}{5}$ (i.e., 0.6).

3) Cosine similarity

Cosine similarity [21], a metric that measures the similarity of two non-zero vectors, is the cosine of the angle between these two vectors in a vector space. These non-zero vectors are obtained by using word embedding, an NLP technique that was mentioned previously. Cosine similarity can be calculated by using an open-source software library SpaCy [22]. When

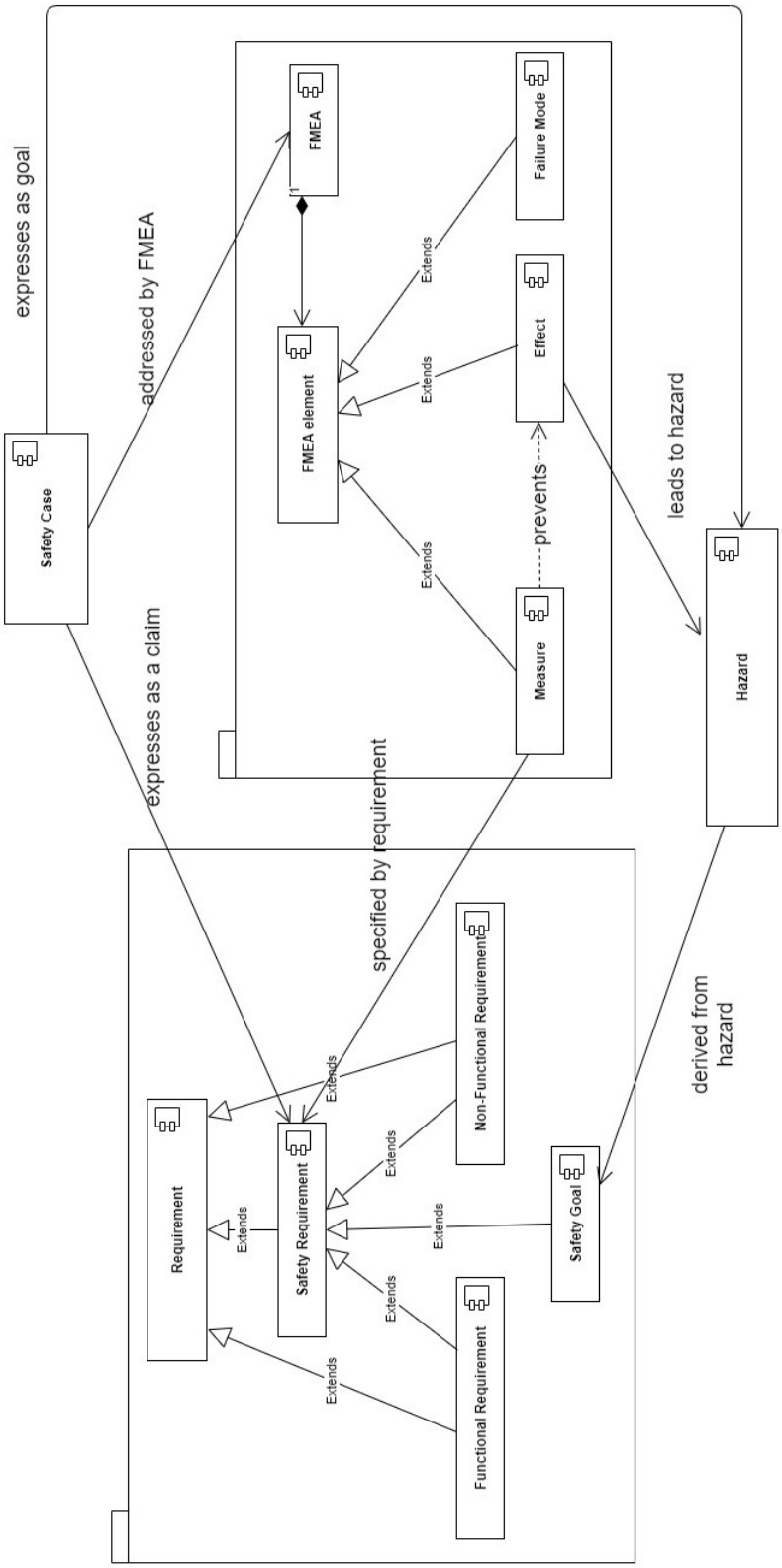

Fig. 1: Information model the TIM is based on

using SpaCy to calculate cosine similarity between two texts, the texts will first be converted to word vectors. Then the cosine similarity between these two word vectors is calculated.

4) Universal sentence encoder similarity

Instead of using word embedding to calculate cosine similarity, sentence embedding [23] can also be used to calculate the similarity between texts. Instead of converting each word to vectors, sentence embedding converts the entire sentence to vectors. One popular technique of sentence embedding is the universal sentence encoder. Universal sentence encoder [23] embeds sentences into high-dimensional vectors, which can also be used for checking similarities between texts.

E. Classification

To evaluate if a metric can be used for inconsistency detection, we need to investigate if the inconsistency classification can be done based on this metric. For detecting inconsistencies between *safety artifacts*, the classification is specifically text classification.

1) Text classification

Text classification, which aims to predict which label or tag a textual unit such as a sentence belongs to, categorizes different texts into organized groups [24]. When implementing text classification within a machine learning system, there are inputs and outputs. These inputs that are fed into the machine learning system are called features. From analyzing the features, the system can generate labels as outputs.

2) Classifier

Minaee et al. [24] mentioned that within machine learning models, features are extracted from the input text, and then are fed to a classifier to predict. The classifier is an algorithm that is trained to give a prediction based on the feature and training data. Two commonly used classifiers were chosen in this research. They are Support Vector Machine (SVM) and K-Nearest Neighbors (KNN). To evaluate the performance of the classifier, there are multiple measures. The measures that were used to evaluate the performance of the classifier are presented in Subsection II-F. Ground truth is needed to evaluate the prediction performance of the classifier, which is the reality that the classifier generates predictions based on. A threshold of similarities needs to be inferred to evaluate if the similarity metric can be used to train a classifier.

3) Threshold

Inferring a threshold can investigate if similarity metrics can be used for inconsistency classification. The threshold can provide guidance on what kind of values will be classified into which class. With the threshold, we can train a classifier and then evaluate how good the classification can be based on the specific metrics. To evaluate the performance of a classifier, the following measures are usually used.

F. Evaluation

To evaluate if different kinds of similarities can help with the consistency classification of texts, different measures can be used for analysis. Besides, to evaluate the classifier, some data from the data that are fed to the classifier need to act as the test set [5]. To increase the accuracy of evaluation, this test set cannot be too small [5]. While increasing the size of the test set can lead to a lack of training data, which can influence the performance of the classifier [5]. Therefore, a technique called cross-validation is introduced to solve this problem.

1) Cross-validation

Cross-validation enables us to evaluate different test sets and then combine their scores [5]. Particularly, we divide the test sets into N subsets, which are called folds [5]. For every fold, the data in other folds act as training data, and then we test the trained model on that fold [5]. The test is repeated N times with training sets composed of those subsets that were not selected as test sets.

2) Precision

The first evaluation metric is precision. Precision illustrates the percentage of examples that truly belong to one class out of the total examples that are predicted to belong to one class, which is based on the prediction results.

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

TP: true positive, e.g., for a binary classifier, a cat picture is predicted as a cat; **FP:** false positive, e.g., a not-cat picture is predicted as a cat; **FN:** false negative, e.g., a cat picture is predicted as a not-cat.

3) Recall

Recall illustrates how many percents of truly positive examples are correctly predicted.

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

4) F-score (F)

Since precision and recall can differ a lot from each other, a combination of precision and recall is needed. F-score is a combination of precision and recall, which gives a single score [5].

$$F = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

III. Related work

The previous works that investigate methods for detecting inconsistencies of the systems can be categorized into two types. One is without involving NLP techniques, the other one is involving NLP techniques to ensure some specific kinds of consistency, such as terminology consistency. There also exist previous works that investigate how NLP handles text analysis.

A. Detecting inconsistencies without NLP

Bidirectional transformations (BX), which have different applications in different domains, serve to maintain consistency among interdependent *artifacts* [25], [26]. One method of checking if BX is correct that Gibbons et al. [26] presented is a relational model. In this model, there is one consistency relation which consists of two model sets and two consistency restorers. BX is stated as "correct" if and only if both consistency restorers indeed complete the consistency restorations. Also, it will be stated as "no harm" if two sets

are already consistent. In this method, the consistency between all related *artifacts* is checked via the transformation, which is represented as a relationship between every two *artifacts* [26]. In this research, I checked the consistency between every two *artifacts* that are connected by a traceability link, which was also considered as a relationship. However, instead of focusing on the consistency restorations, this research focused on identifying the suspected *artifact* links.

Sadeghi et al. [27] used a white-box method to guarantee the consistency of different components of the system. The authors used specific models, including a consistency model, to check the safety adaptation of the system components. When the code can not pass the consistency model's check, the component will be registered as "unsafe". This way, it ensures that the inconsistencies can be detected before they might place the system at risk. Ensuring the consistency between every two traceable components is also what my research focused on. However, this white-box method does not involve NLP to detect inconsistencies. It instead uses intra- and inter-component dependency models to make sure that the system is always in a consistent state [27].

B. Detecting inconsistencies with NLP

According to [28], the authors concluded that applying NLP can decrease the influence of poor modeling practices such as naming inconsistencies or ambiguity. They applied NLP techniques to extract the SBVR (Semantics of Business Vocabulary and Business Rules) business terms from visual models automatically. Besides, by utilizing algorithms that involve modern NLP techniques, the authors managed to improve the quality of their previous designed transformation model for observing the consistency between different business words.

Kim et al. [29] used an NLP parser to develop a Code Dictionary that records domain words with dominant part-of-speech and idiom identifiers, to detect inconsistencies of identifiers. The authors mentioned one challenge of this method is that NLP parsers cannot recognize if one word is abbreviated or not. For example, NLP parsers might not recognize that "doc" means "document", which can decrease the accuracy of the parsed results [29].

These two papers illustrate that inconsistencies between terminologies can be detected automatically by applying Natural language processing techniques. Instead of only checking the terminology consistency, I investigated how NLP can detect the inconsistencies of two textual components which have a traceability link between each other. In order to avoid the challenge mentioned in [29], I made sure that the NLP techniques were involved in the designed artifact can recognize the abbreviated words.

C. NLP used for analyzing textual artifacts

In [10], the authors emphasize the importance of meeting operating systems' safety policies that are written in human languages. Thus they applied NLP to handle the extraction of those safety policies from requirement documents. Safety cases, *safety artifacts*, and other elements that appear during software development are written in natural language. The achievement of automatic analysis on textual contents can enhance not only the reliability and quality of the software but also make work easier when developing and maintaining the system. One challenge for NLP to work as expected, the authors found, is that the text is perhaps vague and not well-structured, or the information contained in the text is not explicit [10].

This literature shows that NLP is a possible method when handling text analysis and its limitation. In our research, NLP techniques were involved for detecting inconsistencies of the textual *safety artifacts* by analyzing the similarity or word vectors of texts within two *safety artifacts*, which proves that NLP is indeed a feasible method when analyzing text. In order to avoid the challenge mentioned in this literature, we compared the inconsistencies that the designed artifact found with the ones we found manually. When conducting the manual annotation, we managed to find texts that are not explicit. According to the characteristics of these texts, we found improvement of our designed artifact, to make the designed artifact better handle vague or not well-structured texts.

IV. Research methodology

This study aims to investigate methods to identify inconsistencies between different traceable textual components that are related to safety cases, by using natural language processing techniques. I broke down the aim into two main research questions. The second main research question included three sub-questions.

RQ1: How can inconsistencies between *safety artifacts* that safety cases rely on be detected with the help of NLP techniques?

RQ2: How accurate are different NLP techniques for detecting inconsistencies between *safety artifacts*?

RQ2a: Does text normalization affect inconsistency detection?

RQ2b: Do different kinds of features affect inconsistency detection?

RQ2c: Is there a difference between different classification algorithms on inconsistency detection? (in particular SVM vs KNN)

RQ1 aimed at investigating if using NLP can help with identifying inconsistencies between texts. RQ2 aimed at evaluating the accuracy of inconsistency detection by applying different NLP techniques. RQ2a aimed at investigating if normalizing texts is beneficial when detecting inconsistencies between textual *safety artifacts*. RQ2b aimed at investigating if different kinds of features affect the inconsistency classification performance. RQ2c aimed at investigating the performance differences between the SVM classifier and the KNN classifier.

Design science research was selected as the research method. Design science research is a problem-solving paradigm that creates and evaluates IT artifacts to solve identified organizational problems in Information Technology [30]. In my case, the artifact that I designed and developed

is a method that uses NLP to detect inconsistency between two *safety artifacts* that are connected with a traceability link within the dataset developed by Maro et al. [6]. What *safety artifacts* my method is applied to and how these *safety artifacts* relate to each other are shown in the following subsection.

I conducted this research by following the design science research process model adapted from Peffers et al. [31]. As shown in Fig. 2, this research was conducted in three iterations. Every iteration includes five steps, which are identify problem and motivate, define objectives of a solution, design and development, demonstration, and evaluation. The final iteration additionally includes one more step, which is communication. The data collected from these three iterations contributed to answering RQ1, RQ2, and RQ3.

In the first iteration, the aim was to develop and evaluate an artifact to investigate a method that can extract the texts of each component of every trace link and calculate the similarity between each effect and its related safety requirements. In the second iteration, the aim was to improve the designed artifact by increasing the performance of the inconsistency detection, based on the evaluation result of the previous iteration. In the third iteration, the aim was to train a classifier to investigate if checking similarities can help with identifying inconsistencies.

A. First iteration

During *design and development* step in the first iteration, firstly, I conducted data preprocessing. As presented in Subsection V-A1, I simplified the original data model of the dataset developed by Maro et al. [6] to understand how I want the artifact to identify inconsistencies between *safety artifacts*. Secondly, the initial version of the artifact was developed, which is a method that detects consistency between *safety artifacts* according to similarity calculated by baseline methods. The method includes an algorithm that can extract the textual data from the dataset, normalize the data, and check the similarity between textual data that are connected by traceability links by using baseline methods. The source code of the algorithm can be found on the GitHub repository [1]. Two baseline methods were utilized within the first iteration to calculate the similarity between each effect-requirement sentence pair according to two basic similarity metrics. One similarity metric is Levenshtein distance, the other one is Jaccard index. I chose these two similarity metrics because these two metrics are basic methods to measure text similarity. In order to investigate if NLP techniques are better than techniques without NLP at identifying text inconsistencies, a baseline method is necessary. Besides, I chose these two similarity metrics because they are either proven to be effective for similar tasks in previous researches. According to [32], Levenshtein distance is proven to be the most promising text distance measure compared to other text distance measures. According to [33], Jaccard index can be used to help detect inconsistent annotations in an annotated medical document. The data normalization includes tokenizing the textual data, removing stop words, removing punctuation, and lowercasing the textual data. Tokenizing transfers the raw text data into a list of words and punctuation [5]. Thirdly, as presented in Subsection V-A2, I conducted manual annotation to identify the consistency of each sub-graph to gather ground truth data for evaluating classifiers that were based on different metrics.

During the *demonstration* step, this algorithm was applied to the dataset developed by Maro et al. [6]. During the *evaluation* step, I inferred the optimal thresholds of similarities calculated by Levenshtein distance and Jaccard index. The source code that was used to infer threshold can be found on the GitHub repository [1]. As described in Section II-E3, thresholds can be used for classification. Inferring the thresholds was a way to investigate if it was possible to train a classifier to achieve consistency classification between *safety artifacts* according to similarities calculated by baseline methods.

B. Second iteration

During the *evaluation* step in the first iteration, since Levenshtein distance and Jaccard index are baseline methods, the results indicated that they are not good enough to be used as similarity metrics for consistency classification. Therefore, the problem of the second iteration was if similarity metrics that involve NLP can be used for consistency classification.

During *design & development* step in the second iteration, I decided to change the baseline methods to more advanced methods that involve NLP for calculating similarity in the method. I edited the algorithm to calculate similarity by involving other more advanced similarity metrics, which are pairwise similarity, cosine similarity, and universal sentence encoder similarity. I chose these metrics because as described in the Background section, they are more advanced metrics that measure similarity based on word vectors. Besides, based on previous researches, these three similarity metrics were proven to have effective performance when handling similar tasks. As illustrated in [34], pairwise similarity can be used for training a classifier and it is proven to be effective. According to [35], cosine similarity is a similarity metric that can determine the similarity level of textual documents. According to [23], the similarity that is calculated by universal sentence encoder, performs much better than other baseline methods. These similarity metrics need to be measured by involving different NLP techniques.

During the *demonstration* step, this algorithm was applied to the same dataset that is used in the first iteration. During the *evaluation* step, firstly, I inferred the optimal thresholds of pairwise similarity, cosine similarity, and similarity calculated by universal sentence encoder. Secondly, a classifier that did consistency classification according to these similarities was trained. Thirdly, evaluation metrics for classification mentioned in Section II, including precision, recall, and f-score, are calculated to evaluate the classification results. These evaluation metrics can indicate if using these similarity metrics can contribute to identifying inconsistencies between *safety artifacts*.

[1]https://github.com/xunih/consistency-of-safety-artifacts

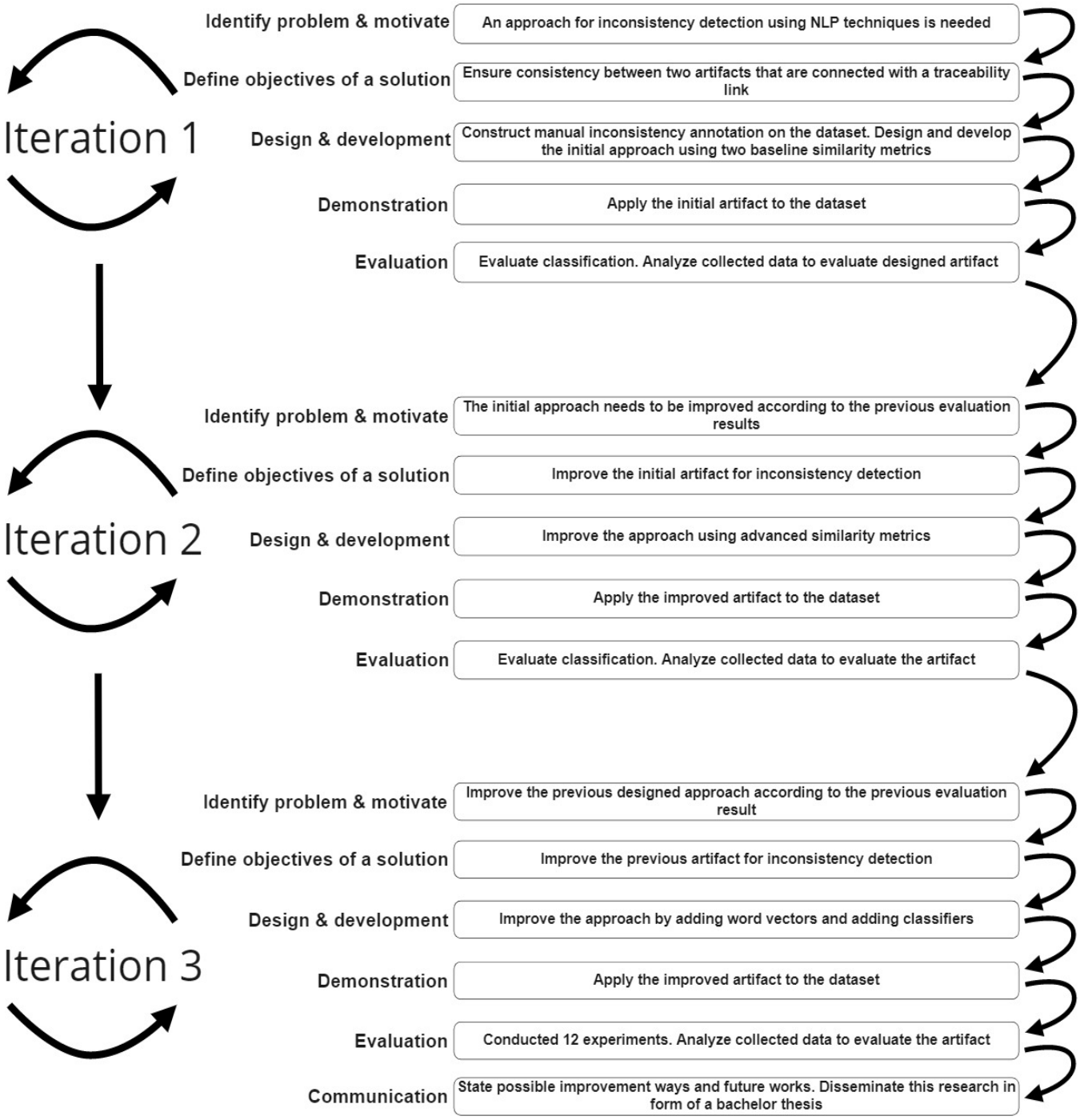

Fig. 2: Design science research process applied to this study

C. Third iteration

During the *evaluation* step in the second iteration, the results indicated that cosine similarity calculated by a large model of SpaCy and similarity calculated by universal sentence encoder may be used for consistency classification. However, the performance of the classification is not good enough. Therefore, the problem of the second iteration was if there exist other features that involve NLP, can perform better than similarity metrics for consistency classification.

During *design & development* step in the third iteration, in the designed artifact, I added two classifiers SVM and KNN that were mentioned in Section II and added features that involve other NLP techniques. These features are not similarities. A KNN classifier was chosen because KNN is one of the oldest, simplest, and most accurate algorithms and KNN can solve many different kinds of classification problems [36]. A SVM classifier was chosen because the SVM classifier has higher accuracy and it is faster when handling nonlinear problems when comparing to other classification algorithms [37]. I added a function to the algorithm that vectorizes the words of the text by using word embedding. As mentioned in Section II, using word embedding can get word vectors of each sentence. The aim of getting these word vectors was to evaluate if word vectors of two sentences can be used for identifying consistency between the two *safety artifacts*.

During the *demonstration* step, this algorithm was applied to the same dataset that is used in the previous iterations. During the *evaluation* step, specific experiments were conducted to evaluate if the classifier that takes similarities as features performed better or the classifier that takes word vectors as features performed better. Besides, I conducted some experiments on the classifier that performs best to evaluate if text normalization affected the inconsistency detection.

D. Validity threats

The validity threats that might affect the research were categorized according the types of validity threats defined in [38].

1) Conclusion validity

Conclusion validity is about if the treatments used in the experiments are related to the outcomes [38]. In this research, experiments were conducted to evaluate the deigned artifact. Choosing statistical test for the data collected from the experiments might affect the conclusion validity. To mitigate the threats to conclusion validity, the normality of the data was checked, in order to decide if parametric tests or non-parametric tests should be chosen.

2) Internal validity

Internal validity is about how sure researchers can be the used treatment led to the outcome [38]. When annotating the consistency between every two linked components manually, the researcher might ignore some inconsistencies. This will affect the evaluation. In order to increase the accuracy of human annotation, I introduced criteria of what kind of texts are consistent and inconsistent. The criteria are illustrated in Section V-A2. The manual annotation was conducted strictly following the criteria. Furthermore, a volunteer was involved to check the annotation work. This volunteer was one academic supervisor of this research.

In addition, the designed artifact was evaluated on one dataset from "MobSTr" [6]. The quality of the dataset might affect the internal validity. This dataset is developed by professional people who have experience working with safety-critical systems. Thus, the quality of this dataset should be good. However, since the quality of this dataset is not evaluated, the outcomes of this research might be affected by the quality of this specific dataset.

3) External validity

External validity is about the generalizaibility of the research results outside the research scope [38]. In this research, the evaluation of the designed artifact was conducted on one dataset from "MobSTr", which was provided by the PANORAMA research project [6]. The generalizability of the designed artifact might be limited because the designed artifact was only examined on one dataset.

V. Results

A. First iteration

1) Data preprocessing

The designed artifact focuses on identifying inconsistencies between *safety artifacts* within the dataset that is presented in Section II. Within the dataset mentioned above, there are multiple effects, which can be related to multiple hazards. Each safety goal has multiple safety requirements. Fig. 3 shows parts of the original data model. To better understand which link the designed artifact should detect inconsistencies on, the original model needed to be simplified. In addition, since within each sub-graph all traceability links are transitive, the sub-graph can be considered as "consistent" as long as the effect is consistent with at least one safety requirement. Therefore, I simplified the data model by considering each sub-graph (e.g., Effect1-Hazard1-Safety Goal1-Safety Requirement1.1) independently. Fig. 4 shows the simplified data model. The artifact was designed to detect the inconsistency of each sub-graph by identifying the inconsistency of each effect-requirement sentence pair within the sub-graph.

2) Manual annotation

The manual annotation was conducted to establish the ground truth, by following the criteria below. The manual annotation results worked as the ground truth data when evaluating different classifier's performance. Ground truth data are a labeled dataset that is large and are assumed accurate in machine learning [39].

Criteria sub-graph consistency: Each sub-graph was annotated as "consistent", "suspect", or "irrelevant". Firstly, for every effect, I went through all safety requirements that the effect is related to, to check if the terminologies used in the *safety artifact* "effect" are used in any of the related safety requirements. Secondly, I checked the semantics of effect and safety requirements, to see if their meanings are similar.

If a safety requirement contains at least an identical or similar term that is used in the effect text, which does not

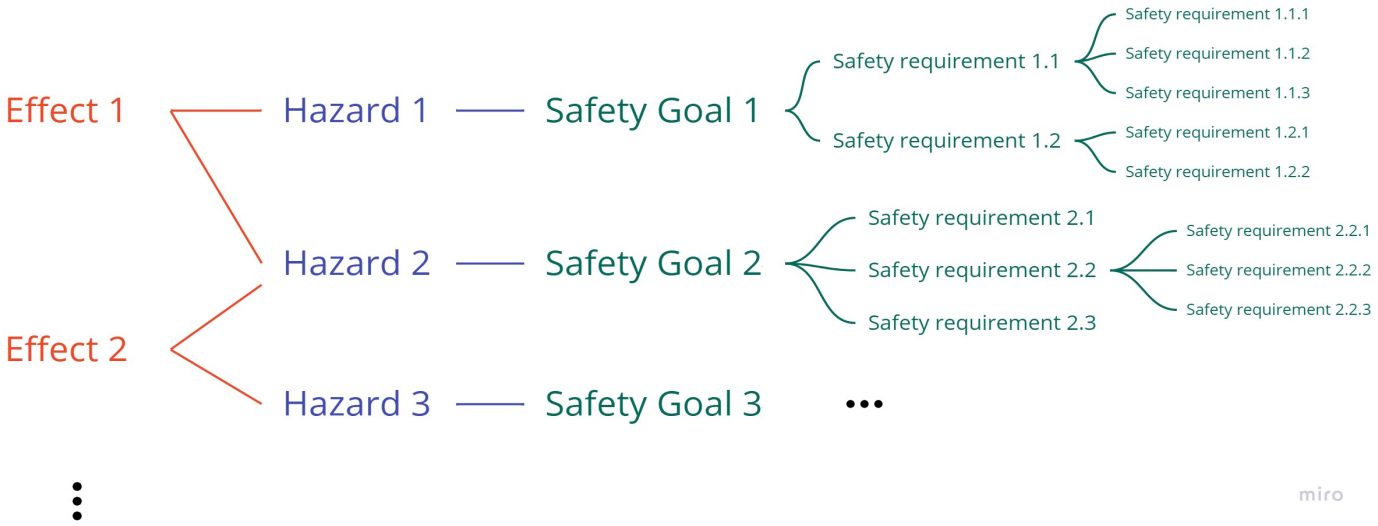

Fig. 3: Original data model

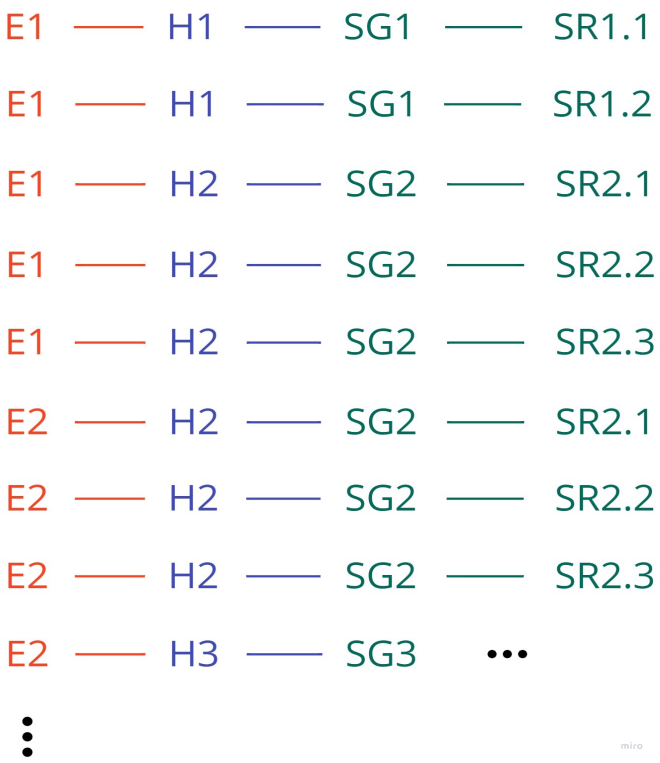

Fig. 4: Detailed simplified data model

belong to stop words in English, and the meaning of the content is relevant to the effect, I annotated this sub-graph as "consistent". If there are no identical or similar terms, and the semantics are not similar, the sub-graph was annotated as "irrelevant". If there is at least one identical or similar term that is used in the effect, but the semantics are not similar, the sub-graph is identified as "suspect".

Table I illustrates examples of different types of sub-graphs that are manually classified. Table II presents the proportions of different kinds of sub-graphs within the dataset.

3) Designed artifact

The initial version of the designed artifact was developed in the first iteration. The designed artifact is a method that uses similarities calculated by Levenshtein distance and Jaccard index between two sentence pairs to detect inconsistencies between *safety artifacts*.

4) Results of evaluation

There were three classes of consistency, which are consistent, suspect, and irrelevant. During the *evaluation* step, I inferred two optimal thresholds **a** and **b** of similarities based on the ground truth introduced in the manual annotation. As mentioned in Section II, thresholds can be used for training a classifier if two thresholds found are not identical and not equal to 0. If the similarity of a sub-graph was smaller than **a**, the sub-graph was classified into the "irrelevant" class. If the similarity of a sub-graph is between **a** and **b**, the sub-graph was classified into the "suspect" class. If the similarity of a sub-graph was larger than **b**, the sub-graph was classified into the "consistent" class.

When inferring the thresholds for similarities that were calculated by Levenshtein distance, both thresholds were 0. This indicates that it was impossible to train a classifier based on the Levenshtein distance similarity. Therefore, calculating Levenshtein distance similarity between *safety artifacts* is not a promising method for identifying inconsistencies.

When inferring the thresholds for similarities that were calculated by Jaccard index, one threshold was 0, the other one was 0.08. Since one threshold was 0, although two thresholds were different, all sub-graphs would be classified into only two classes. This result indicates that it was impossible to train a classifier based on the Jaccard index similarity. Therefore, calculating Jaccard index similarity between *safety artifacts* is also not a promising method for identifying inconsistencies.

Effect	Safety requirement	Classification	Reason
No information about objects provided by the lidar.	The system shall identify when the lidar sensor has failed.	consistent	Same term: 'lidar'. Semantics are relevant
No information about objects provided by the lidar	The system shall use only information from working sensors.	suspect	Same term: 'information'. But semantics not relevant
No information about objects provided by the lidar	The system shall identify sensor failures.	irrelevant	No same/similar terms, semantics not similar

TABLE I: Examples of different kinds of sub-graphs

	Sentence pair	**Proportion**
Consistent	356	24%
Suspect	122	8%
Irrelevant	1010	68%
Total	1488	100%

TABLE II: Proportion of different kinds of sub-graphs

Class	**Precision**	**Recall**	**F-score**
Consistent	0.46	0.49	0.48
Suspect	0.25	0.26	0.26
Irrelevant	0.88	0.35	0.50
Average	0.53	0.37	0.41

TABLE III: Evaluation results of SpaCy cosine similarity classifier (**RQ1, RQ2b**)

B. Second iteration

1) Designed artifact

The designed artifact was improved according to the evaluation results of the first iteration. The improved artifact is a method that uses pairwise similarity, cosine similarity, and universal sentence encoder similarity between two sentence pairs to detect inconsistencies between *safety artifacts*.

2) Results of evaluation

When inferring the thresholds for pairwise similarities, two thresholds were identical. Similarly, when inferring the thresholds for cosine similarities that were calculated by using a small data model of SpaCy, two thresholds were also identical. As mentioned in Section II, SpaCy is an open-source software library for NLP in Python and Cython programming languages. These results indicate that it was impossible to train a classifier based on pairwise similarity or cosine similarity that was calculated by using a small data model of SpaCy.

When inferring the thresholds for cosine similarities that were calculated by using a large data model of SpaCy, two thresholds were not identical and not equal to 0. Therefore, I trained a classifier to do the consistency classification according to these thresholds. I evaluated this classifier by calculating its precision, recall, and f-score. The evaluation results are presented in Table III. From this table, we can see that the average precision and average recall of the classifier are not very high. The precision for the irrelevant class is 87.7%, which is high, but the recall is much lower than precision. While for "consistent" class and "irrelvant" class, precision is similar to recall. Overall, the results indicate that the cosine similarities calculated by the SpaCy large data model are not good enough to be used for consistency classification.

When inferring the thresholds for similarities that were calculated by using universal sentence encoder (USE), two thresholds were not identical and not equal to 0. Therefore, I trained a classifier to do the consistency classification according to these thresholds. I evaluated this classifier by calculating its precision, recall, and f-score. The evaluation results are presented in Table IV. From this table, we can see that the average precision and average recall of the classifier are low, which are all less than 50%. For class "consistent" and "suspect", the recall is much higher than precision. While for "irrelevant" class, the precision is much higher than the recall. Overall the results indicate that the universal sentence encoder similarities are not good enough to be used for consistency classification.

Class	**Precision**	**Recall**	**F-score**
Consistent	0.37	0.83	0.51
Suspect	0.13	0.82	0.10
Irrelevant	0.83	0.50	0.44
Average	0.44	0.47	0.35

TABLE IV: Evaluation results of universal sentence encoder similarity classifier (**RQ1, RQ2b**)

According to Table III and Table IV, we can now answer **RQ1** and **RQ2b**. For **RQ1**, involving some NLP techniques such as universal sentence encoder and SpaCy to calculate the similarity between two texts can help with identifying inconsistencies. Therefore, with the help of NLP techniques, the inconsistencies between textual *safety artifacts* are possible to be identified by checking the similarities. For **RQ2b**, within different features about similarity, the universal sentence encoder performs more accurately than other NLP techniques. When calculating cosine similarity, the results calculated by a large data model of SpaCy are more accurate than a small data model.

C. Third iteration

According to the evaluation results, the performance of the SpaCy cosine similarity classifier and USE classifier is not very good. Therefore, to check if other commonly used classifiers perform better, I added two classifiers SVM and KNN which are commonly used for classification. Besides, to check if other features perform better than similarities, I used word embedding to get the word vectors of each sentence pair. The word vectors were used as features. This aims at

investigating one kind of feature that performs better than similarity for consistency detection.

1) Final designed artifact

In the third iteration, based on the evaluation results of second iteration, I developed the final version of the designed artifact that helps detect inconsistencies between *safety artifacts* of this dataset. The final designed artifact is a method that utilizes word vectors of each effect and requirement sentence pair as features to train a KNN classifier for consistency classification. The sentence pairs within the dataset are not normalized. Fig. 5 presents the designed method. The method includes the following steps: 1. extract each effect and safety requirement sentence pair; 2. conduct manual annotation as ground truth data; 3. apply word embedding to get word vectors of each sentence pair; 4. use word vectors as features to train a KNN classifier with cross-validation. This final designed artifact was evaluated by conducting 12 experiments. The following subsections presents the details of these experiments and their results.

2) Hypotheses

In order to investigate if text normalization will affect the performance of the classifier, I set up hypotheses H_{0A} and H_{1A}. H_{0B} and H_{1B} were set up to test which feature that is fed to the classifier performs best when doing text consistency classification.

H_{0A}: There is no significant difference in the consistency identification between normalized data and non-normalized data.

H_{1A}: There is a significant difference in the consistency identification between normalized data and non-normalized data.

H_{0B}: There is no significant difference in the consistency identification between having only similarities as features, only word vectors as features, and having both as features.

H_{1B}: There is a significant difference in the consistency identification between having only similarities as features, only word vectors as features, and having both as features.

3) Experiments

12 experiments were conducted on the dataset during the *evaluation* step in the third iteration, to evaluate the performance of the classifier. Every experiment consisted of 10-folds. As mentioned in Section II, this is a technique called Cross Validation, which aims to evaluate the prediction performance of the classifier. By evaluating the classifier, firstly we can investigate if consistency classification between textual *safety artifacts* can be identified with the help of NLP. Secondly, we can investigate if text normalization will affect the consistency classification. Thirdly, we can investigate which feature and which classifier performs best. With these investigations, I evaluated if the designed method helped the consistency detection and what combination of different techniques should be within the method to obtain best consistency detection results. Table V shows the experiments and their general outcomes.

According to Table V, we can answer **RQ1**. For **RQ1**, other than the answers mentioned in Section V-B2, using the NLP technique word embedding to obtain the word vectors of the texts and use the word vectors to identify the inconsistencies is also possible. Therefore, with the help of NLP techniques, the inconsistencies between textual *safety artifacts* can be identified converting texts to word vectors.

As shown in Table V and Fig. 6, we can see that in general KNN classifier performs better than the SVM classifier. To test the hypotheses described above, I chose the classifier with better performance, which is KNN. Therefore, experiments 2, 4, 6, 8, 10, and 12 contributed to testing these two hypotheses.

According to Table V and Fig. 6, we can now answer **RQ2c**. For **RQ2c**, according to the performance of the KNN classifier and SVC classifier, there can be a difference between different classification algorithms on inconsistency detection. KNN classifier performs better than the SVM classifier.

Fig. 7 and Fig. 8 illustrate the boxplots of performance of KNN classifier with 10-folds when using text normalization and without using text normalization. We can tell that when using similarities as features, the use of text normalization affects the performance of the KNN classifier. The KNN classifier with text normalization performs better than the KNN classifier without text normalization. However, when involving word vectors as features, the classifier without text normalization performs better than the one with text normalization. The boxplots of the performance of the KNN classifier when using word vectors as features can be found in Fig. 10 in the Appendix.

Fig. 9 illustrate the boxplots of average precision, average recall, and average f-score of KNN classifier with 10-folds when using different features and without text normalization. The boxplots of average precision, average recall, and average f-score of KNN classifier with 10-folds when using different features and with text normalization can be found in Fig. 11 in the Appendix. From these two figures, We can tell that no matter with text normalization or not, involving word vectors as features perform much better than only involving similarities. Besides, when involving word vectors, adding similarities as features does not have a profound impact on the performance of the KNN classifier.

4) Statistical tests

To conduct statistical hypothesis testing for H_{0A} and H_{1A}, firstly, I checked the normality of the sample data. The results showed that data were not normally distributed. Therefore, a non-parametric statistical test was chosen. Secondly, since I would like to compare the results of every two experiments, a statistical test that is used for comparing outcomes between two independent groups of data was chosen. Hence, I chose a non-parametric statistical test *Wilcoxon rank-sum test*. *Wilcoxon rank-sum test* is equivalent to a *Mann-Whitney U-test* [40], which is used to compare outcomes between two independent groups of data. I conducted this test on the outcomes of experiments 2 and 4; experiments 6 and 8; experiments 10 and 12. All of the test results indicated that we failed to reject the null hypothesis with a significance level alpha 0.05. This means that we cannot tell whether text

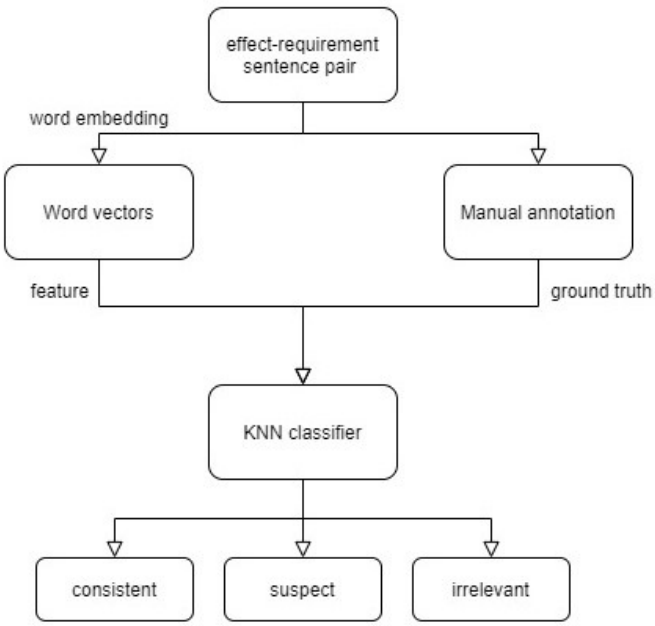

Fig. 5: Final design of artifact

Experiment	Normalization	Feature	Classifier	Precision	Recall	F-score
1	Normalized	Only similarities	SVM	0.50	0.55	0.47
2	Normalized	Only similarities	KNN	0.58	0.49	0.48
3	Non-normalized	Only similarities	SVM	0.39	0.43	0.35
4	Non-normalized	Only similarities	KNN	0.54	0.52	0.50
5	Normalized	Only word vectors	SVM	0.51	0.50	0.48
6	Normalized	Only word vectors	KNN	0.65	0.66	0.59
7	Non-normalized	Only word vectors	SVM	0.54	0.54	0.51
8	Non-normalized	Only word vectors	KNN	**0.74**	**0.71**	**0.68**
9	Normalized	Similarities and word vectors	SVM	0.51	0.51	0.49
10	Normalized	Similarities and word vectors	KNN	0.65	0.66	0.59
11	Non-normalized	Similarities and word vectors	SVM	0.54	0.53	0.51
12	Non-normalized	Similarities and word vectors	KNN	**0.74**	**0.71**	**0.68**

TABLE V: Experiment outcome (**RQ1, RQ2a, RQ2b, RQ2c**). Precision, recall, and F-score are average precision/recall/F-score **Bold** denotes that the best results with a statistical significance 0.05.

normalization has an impact on the performance of the KNN classifier.

Now **RQ2a** can be answered. According to this statistical test result, for **RQ2a**, we cannot tell whether text normalization has an influence on identifying inconsistencies or not. However, according to Table V, Fig. 7, Fig. 8, and Fig. 10, the performance of the KNN and SVM classifiers within the experiments can be different. For different classifiers, text normalization can either improve the performance or reduce the performance.

To conduct statistical hypothesis testing for H_{0B} and H_{1B}, firstly, since the data were not normally distributed, a non-parametric statistical test was chosen. Secondly, since I would like to compare the results of every three experiments, a statistical test that is used for comparing outcomes between three independent groups of data was chosen. I chose *Kruskal-Wallis Test* [41]. *Kruskal-Wallis Test* is a non-parametric statistical test to compare the outcomes between more than 2 independent groups of data. I conducted this test on the outcomes of experiments 2, 6, and 10; experiments 4, 8, and 12. The results indicated that we reject the null hypothesis with a significance level alpha $= 0.05$. This means that the difference between having only similarities as features, only word vectors as features, and having both as features is statistically different.

Now **RQ2b** can be answered. According to the statistical result, there is a statistical difference between having only similarities as features, only word vectors as features, and having both as features. According to Table V, Fig. 9 and Fig. 11, For **RQ2b**, other than the answers mentioned in Section V-B2, involving word embeddings perform more accurately than involving similarity. Therefore, different kinds of features may affect inconsistency detection.

Combining the answers to **RQ2a**, **RQ2b**, and **RQ2c**, now we can answer **RQ2**. Different NLP techniques have different performance when detecting inconsistencies. Using different

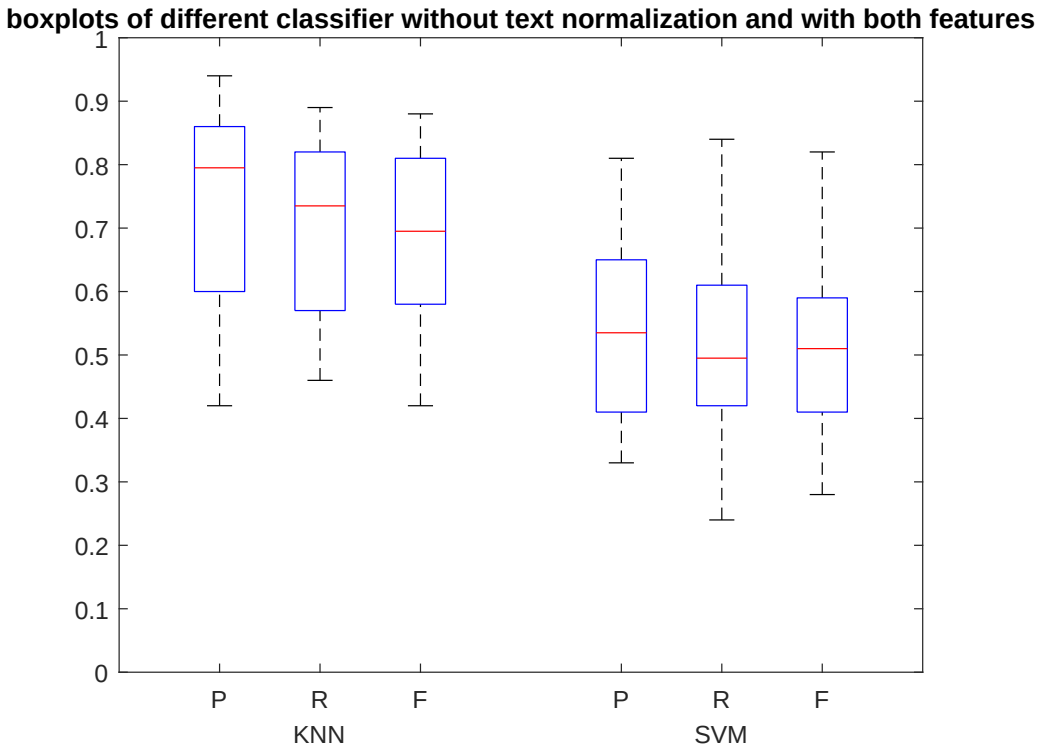

Fig. 6: Boxplots of KNN and SVM classifier (similarities and word vectors as features, without text normalization)(**RQ2c**)

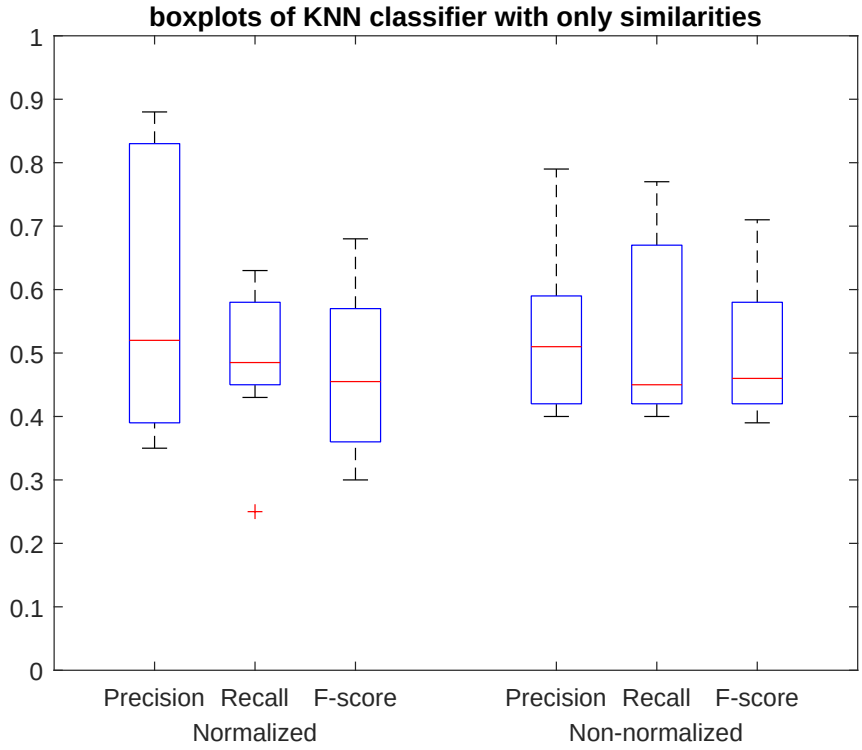

Fig. 7: Boxplots of KNN classifier (similarities as features)(**RQ2a**)

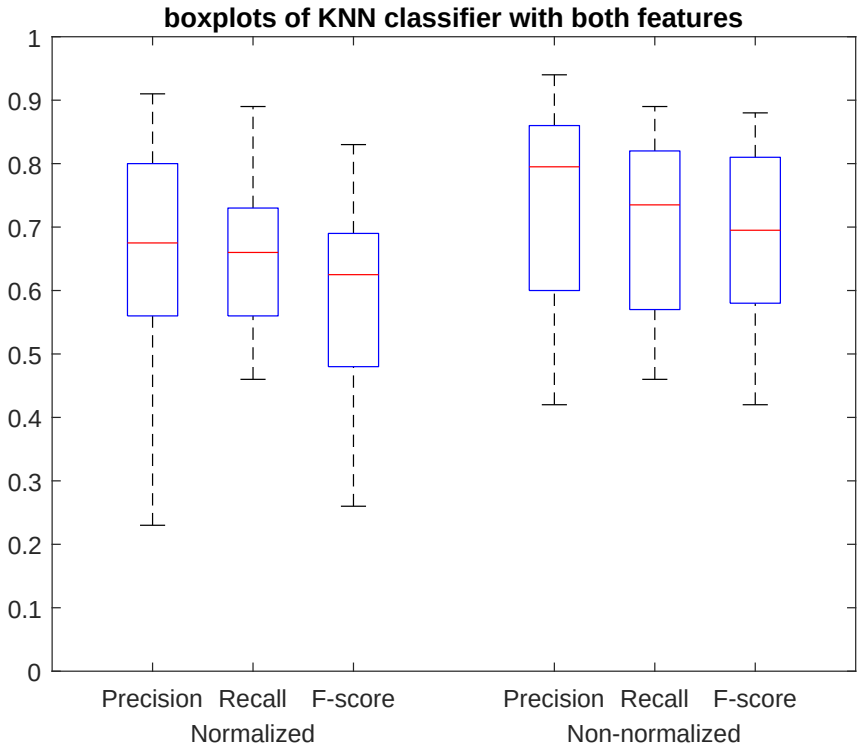

Fig. 8: Boxplots of KNN classifier (both similarities and word vectors as features)(**RQ2a**)

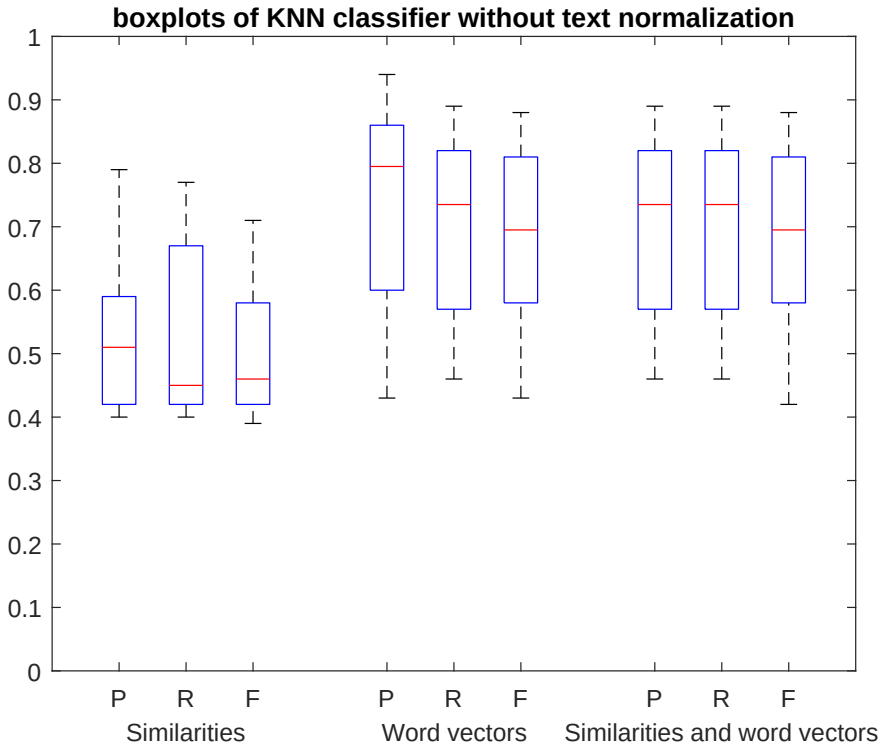

Fig. 9: Boxplots of KNN classifier (without text normalization)(**RQ2b**)

classifiers and different text pre-processing might have an impact on the detection performance. Overall, having word vectors as features that are obtained by NLP technique word embeddings, using KNN classifier, and without using text normalization performs best.

VI. Discussion

According to the answers to RQ1, engineers within the safety-critical systems may utilize NLP techniques to detect inconsistencies between *safety artifacts*. These NLP techniques can be checking similarities of texts or word embedding techniques. By understanding the answers to RQ2a, engineers should be careful with text normalization when detecting inconsistencies between textual *safety artifacts*. For different classifiers, the impact of text normalization might be different. Hence, engineers should consider the impact of text normalization after choosing different classifiers. Based on the answers to RQ2b, engineers should choose features that can lead to best classification results when detecting inconsistencies between *safety artifacts*. Different features to be fed to the classifiers may affect the inconsistency detection. So engineers should be careful with choosing features. According to the answers to RQ2c, when detecting inconsistencies between *safety artifacts*, engineers should choose the classifier that performs best for specific classification problems. Different classifiers might affect inconsistency detection. Concerning the answers to RQ2, engineers within safety-critical systems should choose specific NLP techniques that can perform best for identifying inconsistencies between *safety artifacts*. Different NLP techniques can have an impact on detecting inconsistencies.

As mentioned in Section III, there are models that try to reach a similar goal as this paper's designed artifact. Within the model of Bidirectional transformations [25], [26] and the model developed by Sadeghi et al. [27], the models will mark the consistency state. When the consistency state shows that the system is not consistent, it can warn engineers that there might exist suspected risk within the system. Similarly, in the method presented in this paper, if a traceability link between *safety artifacts* is not marked as "consistent", it can warn engineers that something might be wrong within this link. The difference between this method and the previous two methods is that this method uses the NLP technique word embedding to detect the inconsistencies.

In order to warn engineers that there might exist inconsistencies within some traceability links, the recall is more important for classes "suspect" and "irrelevant". This is because engineers would not want to miss any inconsistent links. However, precision is more important than recall for "consistent" classes. This is because it will not have a bad influence on the system if some consistent links are marked as "suspect" or "irrelevant".

Compared to the NLP techniques used in [28] and [29], the method presented in this paper uses a different NLP technique and is used for a different field. However, the results of this study are in agreement with the findings in [28] and [29] which showed the potential of NLP within detecting inconsistencies.

The results of this study further support the idea of [10], which argues that NLP can be used for text analysis. Although the limitations mentioned in [10] did not show up, this study has its own limitations. Firstly, the manual annotation results themselves contribute to helping detect the inconsistencies between *safety artifacts* within the dataset. However, there was only one annotator for the manual annotation. Thus, inter-annotator agreement [42] cannot be measured. This means that those results might not be reliable enough to act as the ground truth. Although there was a supervisor who volunteered to validate the manual annotation results to try to decrease this risk. Secondly, this method was developed specifically for the dataset presented in Section II-B. Thus, the generalizability of this method cannot be ensured, which means that it might not work as expected for other data models. Thirdly, the quality of the chosen dataset is a limitation. Since quality of the dataset is not evaluated, it might affect the outcomes of this research. Datasets with different qualities and contents might have a different impact on the outcomes.

VII. Conclusion

This paper demonstrates the designed method that assists the inconsistency detection between *safety artifacts* that are connected by traceability links within the MobSTr dataset developed by Maro et al. [6], by involving NLP techniques. This will be beneficial for current safety-critical systems. The practitioners who work with safety-critical systems in the software engineering field can use the designed artifact to help identify inconsistencies in systems. Furthermore, it will provide fundamental information about how NLP can contribute to ensuring consistency between *safety artifacts* that safety cases rely on. The fundamental information is that checking the similarity between two texts and obtaining word vectors can help with inconsistency detection. Having this information, researchers can investigate more about how NLP can help with inconsistency detection. Researchers can improve the designed artifact according to the evaluation results of the final iteration, which will contribute to better inconsistency detection for safety-critical systems. As mentioned in Section VI and Section IV-D3, the designed artifact was only evaluated on one dataset in this research. Therefore, future studies can be trying the designed artifact on more datasets that are from safety-critical systems. Also, future studies can be investigating if other NLP techniques can contribute to detecting inconsistencies within safety-critical systems.

Acknowledgement

I would like to thank my supervisors Jan-Philipp Steghöfer and Peter Ljunglöf for their patient guidance and valuable feedback.

References

[1] A. Paz, G. El Boussaidi, and M. Hafedh, "checsdm: A method for ensuring consistency in heterogeneous safety-critical system design," *IEEE Transactions on Software Engineering*, 2020.

[2] S. Burton and A. Habermann, "Automotive Systems Engineering und Functional Safety: The Way Forward," in *Embedded Real Time Software and Systems (ERTS2012)*, Toulouse, France, Feb. 2012. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02263463

[3] A. Egyed, K. Zeman, P. Hehenberger, and A. Demuth, "Maintaining consistency across engineering artifacts," *Computer*, vol. 51, no. 2, pp. 28–35, 2018.

[4] M. Wirsing and A. Knapp, "View consistency in software development," in *Radical Innovations of Software and Systems Engineering in the Future*, M. Wirsing, A. Knapp, and S. Balsamo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 341–357.

[5] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language Toolkit*. " O'Reilly Media, Inc.", 2009. [Online]. Available: http://www.nltk.org/book/

[6] S. Maro, J.-P. Steghöfer, M. Zeller, and B. P. Koopmann, "Waters 2019 challenge artefacts provided by the panorama project," https://github.com/Salome-Maro/waters_challenge_artifacts, 2020.

[7] J. C. Knight, "Safety critical systems: Challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 547–550. [Online]. Available: https://doi.org/10.1145/581339.581406

[8] Z. Langari and T. Maibaum, "Safety cases: A review of challenges," in *Proceedings of the 1st International Workshop on Assurance Cases for Software-Intensive Systems*, ser. ASSURE '13. IEEE Press, 2013, p. 1–6.

[9] A. Rani and G. Aggarwal, "Advanced practices to detect ambiguities and inconsistencies from software requirements," in *2018 International Conference on System Modeling Advancement in Research Trends (SMART)*, 2018, pp. 17–21.

[10] T. Xie and W. Enck, "Text analytics for security: Tutorial," in *Proceedings of the Symposium and Bootcamp on the Science of Security*, ser. HotSos '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 124–125. [Online]. Available: https://doi.org/10.1145/2898375.2898397

[11] X. Chen, Z. Zhong, Z. Jin, M. Zhang, T. Li, X. Chen, and T. Zhou, "Automating consistency verification of safety requirements for railway interlocking systems," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 2019, pp. 308–318.

[12] A. Lim, "Joo koon collision: 'inadvertent removal' of software fix led to collision," https://www.straitstimes.com/singapore/transport/inadvertent-removal-of-software-fix-led-to-collision, Nov 2017.

[13] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, 2015.

[14] W. H. Gomaa, A. A. Fahmy *et al.*, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, pp. 13–18, 2013.

[15] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 1–14. [Online]. Available: https://www.aclweb.org/anthology/S17-2001

[16] R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards, "Normalization of non-standard words," *Computer Speech & Language*, vol. 15, no. 3, pp. 287–333, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S088523080190169X

[17] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin, "Safety cases and their role in iso 26262 functional safety assessment," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2013, pp. 154–165.

[18] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," in *J. Mach. Learn. Res.*, 2000.

[19] Wikipedia contributors, "Levenshtein distance — Wikipedia, the free encyclopedia," 2021, [Online; accessed 11-May-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=1011098657

[20] ——, "Jaccard index — Wikipedia, the free encyclopedia," 2021, [Online; accessed 11-May-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1021726662

[21] ——, "Cosine similarity — Wikipedia, the free encyclopedia," 2021, [Online; accessed 21-May-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1020457433

[22] ——, "Spacy — Wikipedia, the free encyclopedia," 2021, [Online; accessed 27-May-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SpaCy&oldid=1022760170

[23] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[24] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: A comprehensive review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

[25] C. Bernaschina, E. Falzone, P. Fraternali, and S. L. H. Gonzalez, "The virtual developer: Integrating code generation and manual development with conflict resolution," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3340545

[26] J. Gibbons and P. Stevens, *Bidirectional Transformations: International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures*. Springer, 2018, vol. 9715.

[27] A. Sadeghi, N. Esfahani, and S. Malek, "Ensuring the consistency of adaptation through inter- and intra-component dependency analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 1, May 2017. [Online]. Available: https://doi.org/10.1145/3063385

[28] P. Danenas, T. Skersys, and R. Butleris, "Natural language processing-enhanced extraction of SBVR business vocabularies and business rules from UML use case diagrams," *Data & Knowledge Engineering*, vol. 128, p. 101822, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169023X1930299X

[29] S. Kim and D. Kim, "Automatic identifier inconsistency detection using code dictionary," *Empirical Software Engineering*, vol. 21, no. 2, pp. 565–604, 2016.

[30] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004. [Online]. Available: http://www.jstor.org/stable/25148625

[31] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: https://doi.org/10.2753/MIS0742-1222240302

[32] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[33] D. Mahato, D. Dudhal, D. Revagade, and Y. Bhargava, "A method to detect inconsistent annotations in a medical document using umls," in *Proceedings of the 11th Forum for Information Retrieval Evaluation*, ser. FIRE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 47–51. [Online]. Available: https://doi.org/10.1145/3368567.3368577

[34] H. Bao, G. Niu, and M. Sugiyama, "Classification from pairwise similarity and unlabeled data," in *International Conference on Machine Learning*. PMLR, 2018, pp. 452–461.

[35] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, "Cosine similarity to determine similarity measure: Study case in online essay assessment," in *2016 4th International Conference on Cyber and IT Service Management*, 2016, pp. 1–6.

[36] V. Prasath, H. A. A. Alfeilat, A. Hassanat, O. Lasassmeh, A. S. Tarawneh, M. B. Alhasanat, and H. S. E. Salman, "Distance and similarity measures effect on the performance of k-nearest neighbor classifier–a review," *arXiv preprint arXiv:1708.04321*, 2017.

[37] A. Zendehboudi, M. Baseer, and R. Saidur, "Application of support vector machine models for forecasting solar and wind energy resources: A review," *Journal of Cleaner Production*, vol. 199, pp. 272–285, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095965261832153X

[38] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey." in *Seke*, 2010, pp. 374–379.

[39] N. Zhou, Z. D. Siegel, S. Zarecor, N. Lee, D. A. Campbell, C. M. Andorf, D. Nettleton, C. J. Lawrence-Dill, B. Ganapathysubramanian, J. W. Kelly *et al.*, "Crowdsourcing image analysis for plant phenomics to generate ground truth data for machine learning," *PLoS computational biology*, vol. 14, no. 7, p. e1006337, 2018.

[40] T. W. MacFarland and J. M. Yates, "Mann–whitney u test," in *Introduction to nonparametric statistics for the biological sciences using R*. Springer, 2016, pp. 103–132.

[41] P. E. McKight and J. Najab, "Kruskal-wallis test," *The corsini encyclopedia of psychology*, pp. 1–1, 2010.

[42] Wikipedia contributors, "Inter-rater reliability — Wikipedia, the free encyclopedia," 2021, [Online; accessed 30-May-2021]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Inter-rater_reliability&oldid=1018594573

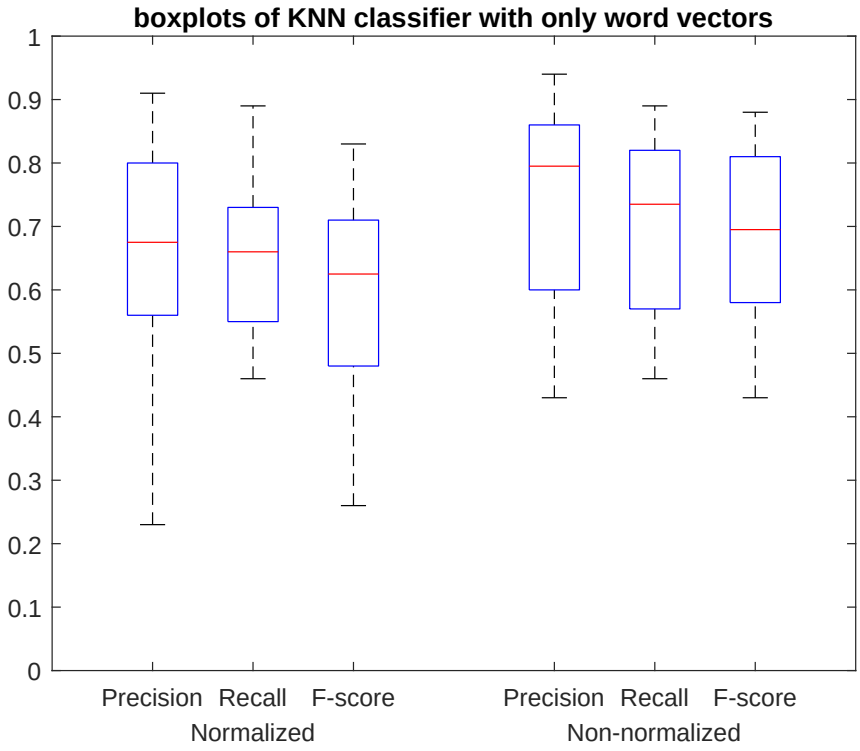

Fig. 10: Boxplots of KNN classifier (word vectors as features)(**RQ2a**)

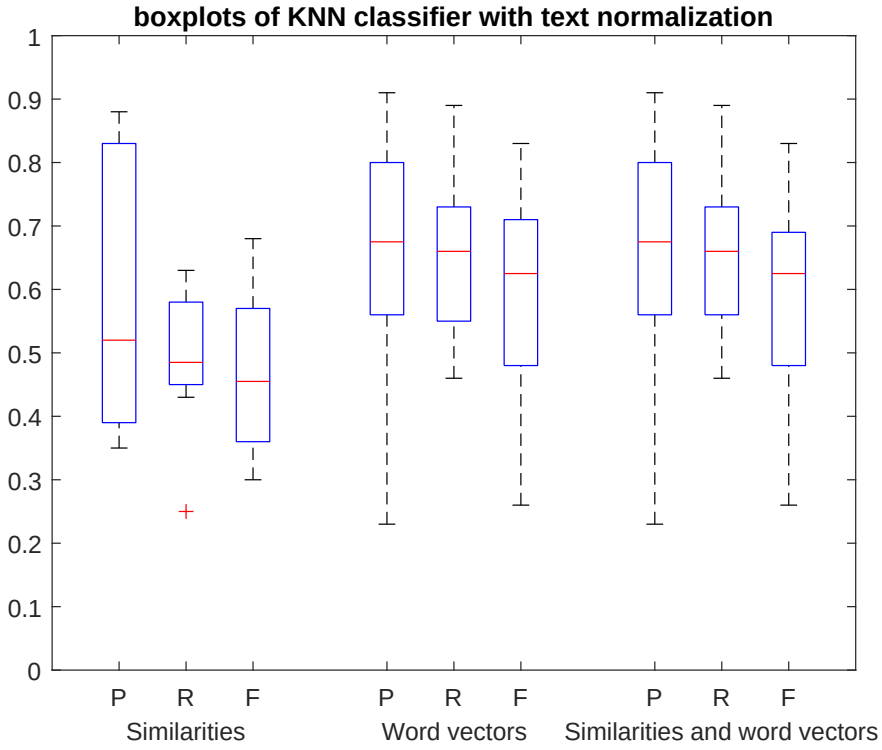

Fig. 11: Boxplots of KNN classifier (with text normalization)(**RQ2b**)