Assignment on Requirements Traceability

D. Barenholz (0998941) J. Versteegen (1018019)

December 19, 2021

1 Introduction

It is not unknown to developers that clients are oftentimes uncertain of the final version of the product they wish to see created. Consequently, the initial requirements that developers start with may very well change over time. If a single requirement changes, it is highly unlikely that this change abstains from impacting the rest of the artefact to be created. Realistically, if some requirement changes, it usually propagates this change through to other requirements, which in turn do the same. However, in reality, developers may not know which requirements affect other requirements. This is the traceability problem: "If requirement A changes, which other requirements should co-evolve?"

A possible solution to the traceability problem is the use of a traceability table or matrix, each with their advantages and disadvantages that we will not discuss here. In the remainder of this report, we simply use 'matrix' to refer to either a traceability table or matrix. Creating such matrices can be done manually, or automatically. Doing this manually, however, requires requirement engineers or otherwise capable people to invest time. Furthermore, it is a cumbersome process that requires one to read all requirements and subsequently understand the entirety of the artefact to then derive traceability between requirements. As artefacts grow in size and complexity, this seems like an insurmountable task. Besides this, artefacts may evolve while one is creating the traceability matrix, which requires them to adapt to this as well. Clearly, this situation is far from ideal.

In this report, we focus on an automatic keyword-based approach proposed by Huffman et. al [3]. This approach, as well as our own keyword-based approach, is described in Section 2. The dataset on which the tool is evaluated is elaborated on in Section 3. The evaluation itself can be found in Section 4. Finally, a discussion on requirement traceability and a conclusion is presented in Sections 5 and 6 respectively.

2 Trace-link Detection Tool

The keyword-based approach that we investigate is commonly known as a 'Vector Space Model' (VSM) algorithm due to the fact that it creates vector representations of each requirement. Based on these vectors it computes a similarity matrix that it uses to generate links between requirements. A concise representation of the specific VSM algorithm we considered, with separate CSV files for high- and low-level requirements, is as follows: 1. Each high- and low-level requirement is parsed and tokenised. 2. Stop-words are detected and removed from the token stream. 3. The remaining tokens are stemmed. 4. The master vocabulary of the repository is constructed. 5. For each requirement, its vector representation is created and stored. 6. A similarity matrix is constructed. 7. Based on the similarity matrix, trace-links are generated.

In Sections 2.1, 2.2, and 2.3 the theoretical aspect of the VSM algorithm is explained in further detail, whereas Section 2.4 shows the implementation. It is worth noting that our own approach builds on the *baseline* VSM approaches we present in Section 2.3.

2.1 Pre-processing steps

Parsing, tokenisation, stop-word filtering, and stemming are well-known in the Natural Language Processing (NLP) community. Parsing is the act of reading textual files and converting them to variables that can be used in a programming language. Tokenisation, then, converts an entire sentence r into separate tokens $\langle t_0, t_1, \ldots \rangle$. From this list of tokens, stop-words (words that are not useful for the purposes of information retrieval such as "shall" and "the") are removed, resulting in a different list of tokens $\langle t'_0, t'_1, \ldots \rangle$. Finally, each token t'_i is brought back to its base form t''_i by means of stemming, to ensure that different forms of the same word are treated as one term. To avoid cumbersome notation, we refer to t''_i as w_i in the upcoming sections.

2.2 Creating the Vector Representation

After pre-processing all requirements into stemmed tokens, we create the vector representation that gives VSM its name. This is done by collecting all tokens w_i into a large set, say M. We denote the size of this set with N. Given M, we construct a vector $r_v = \langle w_0, \ldots, w_N \rangle$ of length N for each requirement r. Each element w_i of some r_v is defined as 0 if M_i does not appear in corresponding requirement r. If it *does* appear, then $w_i = tf \cdot idf$, where tf is the term-frequency (how often M_i occurs in r), and $idf = \log_2 \frac{n}{d}$ is the inverse document frequency (where n is the total number of requirements and d is

the number of requirements containing M_i). Note how this way of creating vector representations implies that all of these vectors, regardless of the requirement it belongs to, have equal dimensions, which is crucial when constructing a similarity matrix.

2.3 Link Generation

Having a vector representation for each requirement, a similarity matrix S is constructed using a measure of similarity (or distance) of choice. In the VSM algorithm, this is the cosine-similarity measure. Hence, $S_{(i,j)}$ contains the cosine-similarity value of vectors r_v corresponding with high-level requirements r_i and low-level requirements r_j. Based on S one can generate links in various ways.

Baseline Approaches:

0. For each r_i, report all r_j with $S_{(i,j)} > 0$.
1. For each r_i, report all r_j with $S_{(i,j)} \geq 0.25$.
2. For each r_i, find maximal $r_{j_{\max}}$ in $S_{(i)}$, then report all r'_j with $S_{(i,j')} \geq 0.67 \cdot S_{(i,j_{\max})}$.

Our Approach:

3. For each r_i, find maximal $r_{j_{\max}}$ in $S_{(i)}$, then report all r'_j with $S_{(i,j')} \geq 0.67 \cdot S_{(i,j_{\max})}$ given that $r_{j_{\max}} \geq 0.25$.

The reason for choosing above approach that we call 'our approach' is due to the fact that it combines two baseline approaches, described earlier as approach one and two, together. You could interpret this as a better performing extension to the second baseline approach.

2.4 Implementation

We present an implementation in Python based on only the `NLTK` [1] toolkit and the well-known `numpy` package, that can be run by means of Docker. The architecture and structure of the code is fairly simple and straightforward. First off, a pure Python IO module named `input_output.py` provides functionality for reading from, and writing to, a CSV file. Note that the extension of the file does not matter, as long as it contains correctly formatted requirements. Secondly, methods for pre-processing, as mentioned in Section 2.1, are available in `preprocessing.py`, a pre-processing module. In order to write a readable version of the VSM algorithm, computations such as tf and idf, as well as the cosine similarity measure, are abstracted away using a helper module, `helpers.py`, containing all

necessary functionality. The VSM algorithm itself is written in the `entry_script.py` file, which is the starting point for the Docker image. We now present said VSM algorithm in detail, in the same order as done earlier: pre-processing, creating the vector representation, and link generation. We conclude this section with a small part on building and running the Docker image.

Pre-processing steps

The pre-processing steps as mentioned in Section 2.1 are abstracted away using a wrapper function: `preprocess()`.

```
# Preprocess the CSV file
low_level = preprocess(low_csv_in)    # dict({id : [word, word, ...]})
high_level = preprocess(high_csv_in) # dict({id : [word, word, ...]})
```

Internally, NLTKs `word_tokenize()` and `PorterStemmer` functionalities are used for tokenisation and stemming respectively. Removing stop-words is done by filtering out all tokens that are contained in a text-based file, `stopwords.txt`. Note that care is taken in the tokenisation method to check for correct parameters; it expects a single requirement to be tokenised, as opposed to a list of requirements. This is the only real robustness check in the entirety of the code base, as it is simply written for a school assignment. As requirements of school assignments do not evolve, this tool is considered an S-type [4] system: there will be no evolution of the code. Hence, ultimately, robustness is not necessary.

The pre-processing method returns a dictionary for each file it is called on. An alternative implementation is to have a function that takes in a list of files, as opposed to only one file. The downside, however, of said alternative, is that the developer loses control over the variable names, which in our code nicely represent the types of requirements that have been pre-processed. Alternatives to NLTKs provided functionality, such as CoreNLP [5] and spaCy [2], could be used, but in the scope of this assignment the differences are negligible. Finally, the reason for choosing a text-based stop-word filtering approach is due to the ease of use: if a stop-word needs to be added (or removed), then this can be done by simply editing the text file. Other approaches, such as NLTKs `corpus.stopwords.words('english')`, require programmatic edits to the set of stop-words, and may not provide clear inspection of said set, whereas a text-based file does provide this.

Creating the Vector Representation

Given the pre-processed requirements, we create the master vocabulary M. In the code, this is denoted by `master_vocab`.

```
# Retrieve the master vocabulary
master_vocab = set(retrieve_master_vocab(low_level, high_level))
```

As can be seen in the code above, we use the `retrieve_master_vocab` method that extracts the correct tokens, given any number of dictionaries. This method could, potentially, be copied to other Python projects to flatten a dictionary of lists. Furthermore, note how M is converted into a set. This is required, as the method used for extracting tokens merely returns a list, whereas a set is required for the master vocabulary. An alternative implementation of retrieving the master vocabulary is a simpler version that only takes two dictionaries; the reason we opted for this slightly more versatile version is due to the extremely low effort necessary, without sacrificing anything in terms of readability. It, thus, is a nicer piece of code and superior in possibly every way. Naturally, we could have taken it a step further; we could have implemented a method that flattens any sort of collection into a list of its basic parts. This, however, is not necessary for the assignment, and introduces more complexity to the code, let alone the fact that it would make the method a lot less readable.

Given M, we need to perform an intermediate step not listed in the theoretical discussion, namely, combining the two dictionaries for low- and high-level requirements into one. Since Python 3.5, there is an idiomatic way to this. As such, we have opted for said idiomatic way.

```
# Create requirements dictionary
requirements = {**low_level, **high_level}
```

As the tool is provided with a *Dockerfile*, ensuring that Python 3.5 is always available, there is no need to consider any compatibility problems.

The last step to creating the vector representation for the VSM algorithm is abstracted away by means of a single method call.

```
# Create vector representation for requirements
vectors = get_vector_representation(master_vocab, requirements)
```

Internally, the `get_vector_representation` method firstly pre-computes each d needed for the idf term as described in Section 2.1, and then *updates* the requirements dictionary: no second dictionary or list is created. Instead, for each key, its values are updated from a list of tokens w_i to a list of numbers r_v. As alternative, we could return a separate dictionary. However, since we still have the original high- and low-level dictionaries, this seems like a waste of memory to us: all possible lookups can still be done since the original dictionaries are still available.

Link Generation

Given a dictionary of the vector representations of the requirements, we create the similarity matrix.

```
# Compute similarity matrix
similarity = compute_similarity_matrix(high_level, low_level, vectors)
```

Similar to previous code, computing the similarity matrix, too, is done by abstracting away from the details: a single method call returns the entire matrix S. Note, however, that it requires both the high- and low-level dictionaries, as opposed to only the vectors. Since this method internally calls a separate, private, method for computing cosine similarities between two arrays, the need arises to correctly index the created vectors which requires correct retrieval of the key of a specific vector. Whereas this may not be the best implementation in terms of speed or memory, it is a very readable one. Alternatively, we could create separate dictionaries for high- and low-level requirements. This essentially splits `vectors` into two dictionaries, and removes the need for passing both the high- and low-level dictionaries. However, this alternative only creates more variables, or replaces the dictionaries, meaning that they could not be used in a different part of the tool. For instance, retrieving the tokens used in a specific requirement is trivial in our implementation, whereas in the alternative this requires a reverse-lookup using M, the master vocabulary.

Based on the similarity matrix, links between requirements are generated.

```
# Link requirements based on match_type
linked_requirements = get_linked_requirements(similarity, high_level, low_level)
```

The same observations as for computing the similarity matrix can be made: firstly, the internal workings of retrieving linked requirements is abstracted away from. Secondly, the

method requires the original dictionaries of requirements. Similar arguments can be made for similar alternative implementations.

```
links_output = r'/output/links.csv'
write_csv(links_output, linked_requirements)
```

Naturally, the linked requirements are outputted to a CSV file such that they can be inspected.

Building & Running

The command for building a Docker image for the tool[1], as well as commands for running said image on Windows (Powershell) and Linux can be found below. Note that the running commands include an x as placeholder for the type of matching to use. This should be an integer number between and including 0 and 3. The specific number refers to the link generation as stated in Section 2.3.

```
# Build the image.
docker build -t 2imp25-assignment-1 ./

# Run the image (Powershell).
docker run --rm -v "$pwd\dataset-1:/input" -v "$pwd\output:/output" 2imp25-assignment-1 x

# Run the image (Linux).
docker run --rm -v "$PWD/dataset-1:/input" -v "$PWD/output:/output" 2imp25-assignment-1 x
```

These commands should be run in the root of the accompanying ZIP file.

3 Waterloo Dataset Description

To evaluate both the baseline approaches as well as our own approach, we use 2 of the 24 publicly available Waterloo datasets. These datasets are commonly used for testing requirement traceability retrieval, and all describe variants of the same system: an IP-telephony service. Both datasets have high- and low-level requirements, where the high-level requirements are features of the system, and the low-level requirements are use cases.

[1] The tool contains evaluation methods that run automatically, and outputs its results as used in Section 4 to the terminal used for running the Docker image.

The features, or *functional requirements*, describe what the system should be able to do. The use cases on the other hand describe a scenario of a user utilising the defined features of the system.

The first dataset, which is group 1, has 49 high-level requirements and 26 low-level requirements whereas the second dataset, group 9, has 85 high-level requirements and 22 low-level requirements. The use cases in dataset 1 show high variance in terms of requirement length, whereas this measure is more equally distributed in dataset 2: dataset 1 has 5 use cases that use 3 words or less and 9 use cases that use 50 words or more, whilst use cases in dataset 2 only have a minimum of 14 words and a maximum of 31 words.

4 Evaluation

In this section we show the results of the tool as described in Section 2: Sections 4.1, 4.2, 4.3, and 4.4 show the results of the different generation approaches in the same order as presented in Section 2.3. Note that each confusion matrix contains a reference to the complete table(s) of misclassified items.

4.1 Approach 0: No filtering

The exact corresponding approach is listed again for convenience:

For each r_i, report all r_j with $S_{(i,j)} > 0$.

We now proceed to evaluate said rule on the two datasets as described in Section 3.

4.1.1 Dataset 1

	predicted	not predicted
actual	38	1
not actual	1083	152

Table 1: Confusion Matrix on Dataset 1, using approach 0.

See also Table 11.

The confusion matrix of the current approach for this dataset can be found on the left, in Table 1. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 38/1121 \approx 0.03$

Recall: $r = 38/39 \approx 0.97$

F_1-score: $2 \cdot \frac{0.03 \cdot 0.97}{0.03 + 0.97} \approx 0.07$

4.1.2 Dataset 2

The confusion matrix of the current approach for this dataset can be found on the right, in Table 2. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

	predicted	not predicted
actual	87	1
not actual	1651	21

Table 2: Confusion Matrix on Dataset 2, using approach 0

See also Tables 15 and 16.

Precision: $p = 87/1738 \approx 0.05$

Recall: $r = 87/88 \approx 0.99$

F_1-score: $2 \cdot \frac{0.05 \cdot 0.99}{0.05 + 0.99} \approx 0.10$

4.1.3 Explanation

As indicated by the extremely high average recall of 0.98, this method predicts almost all links made by experts, with the exception of the link between *F10* and *UC14* in dataset 1, and *F70* and *UC7* in dataset 2. These errors are highly likely due to the fact that we filter out the only possible links between these requirements, respectively the words 'number' and 'of', as these are present in the set of stop-words. On the other hand, the low average precision of 0.04 shows that this approach does not perform as well as one may expect when only looking at recall: it simply returns numerous incorrect links that were not present. This is due to the fact that any similarity in word usage will result in a link being predicted, which is trivially bound to produce exceedingly many false positives.

4.2 Approach 1: Low threshold filtering (≥ 0.25)

The exact corresponding approach is listed again for convenience:

For each r_i, report all r_j with $S_{(i,j)} \geq 0.25$.

We now proceed to evaluate said rule on the two datasets as described in Section 3.

4.2.1 Dataset 1

	predicted	not predicted
actual	20	19
not actual	48	1187

Table 3: Confusion Matrix on Dataset 1, using approach 1

See also Table 12.

The confusion matrix of the current approach for this dataset can be found on the left, in Table 3. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 20/68 \approx 0.29$

Recall: $r = 20/39 \approx 0.51$

F_1-score: $2 \cdot \frac{0.29 \cdot 0.51}{0.29 + 0.51} \approx 0.37$

4.2.2 Dataset 2

The confusion matrix of the current approach for this dataset can be found on the right, in Table 4. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 21/38 \approx 0.55$

Recall: $r = 21/88 \approx 0.24$

F_1-score: $2 \cdot \frac{0.24 \cdot 0.55}{0.24 + 0.55} \approx 0.33$

	predicted	not predicted
actual	21	67
not actual	17	1655

Table 4: Confusion Matrix on Dataset 2, using approach 1

See also Tables 17 and 18.

4.2.3 Explanation

Notice how the precision and recall values over the two different datasets seem to be 'inverted': for the first dataset we see respectively low and high precision and recall, whereas for the second dataset we see respectively high and low precision and recall. From this we can conclude that either this approach is not stable (as in, its performance depends heavily on its input data), or that we need better measures to capture the usability of the approach. Indeed, the F_1-score is relatively equal for both datasets, albeit lower than desired. The typical mistakes of the approach, as mentioned earlier, depend heavily on its input data.

4.3 Approach 2: Filtering based on maximal similarity ($\geq 0.67 \max$)

The exact corresponding approach is listed again for convenience:

For each r_i, find maximal $r_{j_{\max}}$ in $S_{(i)}$, then report all r'_j with $S_{(i,j')} \geq 0.67 \cdot S_{(i,j_{\max})}$.

We now proceed to evaluate said rule on the two datasets as described in Section 3.

4.3.1 Dataset 1

	predicted	not predicted
actual	21	18
not actual	52	1183

Table 5: Confusion Matrix on Dataset 1, using approach 2

See also Table 13.

The confusion matrix of the current approach for this dataset can be found on the left, in Table 5. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 21/73 \approx 0.29$

Recall: $r = 21/39 \approx 0.54$

F_1-score: $2 \cdot \frac{0.29 \cdot 0.54}{0.29 + 0.54} \approx 0.38$

4.3.2 Dataset 2

The confusion matrix of the current approach for this dataset can be found on the right, in Table 6. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 41/282 \approx 0.15$

Recall: $r = 41/88 \approx 0.47$

F_1-score: $2 \cdot \frac{0.15 \cdot 0.47}{0.15 + 0.47} \approx 0.22$

	predicted	not predicted
actual	41	47
not actual	241	1431

Table 6: Confusion Matrix on Dataset 2, using approach 2

See also Table 19.

4.3.3 Explanation

Whilst precision, recall, and the F_1-scores are all relatively low, it is interesting to note that this approach performs considerably better on the dataset 1 as opposed to dataset 2. As

mentioned in Section 3, dataset 1 has high variance in terms of requirement length, hence we initially expect this to be problematic. These results, however, indicate otherwise. This might be due to the fact that having more words implies more possibilities to find links, though we cannot offer a correct explanation.

4.4 Approach 3: Our Approach

The exact corresponding approach is listed again for convenience:

For each r_i, find maximal $r_{j_{\max}}$ in $S_{(i)}$, then report all r'_j with $S_{(i,j')} \geq 0.67 \cdot S_{(i,j_{\max})}$ given that $r_{j_{\max}} \geq 0.25$.

We now proceed to evaluate said rule on the two datasets as described in Section 3.

4.4.1 Dataset 1

	predicted	not predicted
actual	18	21
not actual	30	1205

Table 7: Confusion Matrix on Dataset 1, using approach 3

See also Table 14.

The confusion matrix of the current approach for this dataset can be found on the left, in Table 7. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 18/48 \approx 0.38$

Recall: $r = 18/39 \approx 0.46$

F_1-score: $2 \cdot \frac{0.38 \cdot 0.46}{0.38 + 0.46} \approx 0.41$

4.4.2 Dataset 2

The confusion matrix of the current approach for this dataset can be found on the right, in Table 8. Based on this table, we can compute the precision (p), recall (r), and the F-measure, commonly known as the F_1-score.

Precision: $p = 21/33 \approx 0.64$

Recall: $r = 21/88 \approx 0.24$

F_1-score: $2 \cdot \frac{0.64 \cdot 0.24}{0.64 + 0.24} \approx 0.35$

	predicted	not predicted
actual	21	67
not actual	12	1660

Table 8: Confusion Matrix on Dataset 2, using approach 3

See also Tables 20 and 21.

4.4.3 Explanation

Our approach is a combination of other approaches, which is reflected in the results; The precision and recall values over the two different datasets seem to be 'inverted': for the first dataset we see respectively low and high precision and recall, whereas for the second dataset we see respectively high and low precision and recall. From this we can conclude that either this approach is not stable (as in, its performance depends heavily on its input data), or that we need better measures to capture the usability of the approach. Indeed, the F_1-score is relatively equal for both datasets, albeit lower than desired. The typical mistakes of the approach, as mentioned earlier, depend heavily on its input data.

4.5 Approach Comparison

Each method has already been discussed in its respective section. For convenience sake, an overview of the measures of each approach on the 2 datasets is listed in Table 9 and 10 respectively.

	> 0	≥ 0.25	$\geq 0.67 \max$	Own
Recall	0.97	0.51	0.54	0.46
Precision	0.03	0.29	0.29	0.38
F-Measure	0.07	0.37	0.38	0.41

Table 9: Overview for Dataset 1

	> 0	≥ 0.25	$\geq 0.67 \max$	Own
Recall	0.99	0.24	0.47	0.24
Precision	0.05	0.55	0.15	0.64
F-Measure	0.10	0.33	0.22	0.35

Table 10: Overview for Dataset 2

As can be seen, our approach consistently achieves highest F_1-score. As such, when considering the general applicability of these approaches, our approach is considered to be the best. Should recall be more important, however, then one should use the first approach as evaluated in Section 4.1 with a slight change to the main algorithm, namely, without removing stop words. This will give highest possible recall. It could be argued whether such a case is realistic. Should precision be the most important, then again our approach

surpasses the others. In conclusion, the best approach, from those considered, is our own approach, although its F_1-score is still too low for consistent results. Section 5 elaborates on this problem.

5 Discussion

5.1 Problems

The average F_1-score of all approaches that use some sort of filtering equals to 0.34. Given that this measure ranges from 0 to 1, with 1 being the best, this comes out on the low end. Even the highest achieved F_1-score of 0.41 is less than a half. Clearly, there is some kind of problem with these approaches. Since we do not know the true problem as to why these approaches do not give higher F_1-scores, we *can* propose some ideas as to why the problem we try to solve is difficult.

Firstly, note how the VSM algorithm uses pre-processing. More precisely, note how it filters stop-words from a token stream. If all words present in the requirements are contained in the set of stop-words, no words remain, and the master vocabulary is empty, resulting in no links between requirements whatsoever. A less extreme case of this phenomenon can be seen in approach 0. Said approach, when evaluated on dataset 1, misses the link between high-level requirement *F10* and low-level requirement *UC14*. When inspecting the textual requirements themselves, and comparing them to the stop-words, we see that the only *linking word* between these requirements is 'number'. This word, however, is contained within our set of stop-words to avoid linking requirements that contain 'number of'. As a result, discovering the link between these requirements is impossible for the algorithm. It is, thus, important that the set of stop-words is set correctly, which is an entirely new problem on its own.

Another issue with these approaches is the fact that they are biased: they assume that requirements that are linked together have words in common. This assumption does not necessarily need to hold, falsifying the entire basis of the algorithm. A related issue is that requirements themselves may contain human error. To illustrate this one only has to look at *F10* from dataset 1 and one can see the spelling mistake: **an number**. Whilst the pre-processing step of the VSM algorithm *should*, in theory, solves these minor errors, it is still a possible reason for the relatively bad performance.

As a final note, we have seen in Section 4.3 that the variance of the length of the requirements may influence the performance of an algorithm that must generate links between

said requirements. These issues (what stop-words to choose, how to ensure the mentioned assumption holds, possible human errors, and variability in requirement length) can all influence the performance.

5.2 Improvements

As hinted at in Section 5.1, it seems that there is a fundamental issue with the VSM algorithm as is. Whilst it is academically interesting whether or not this issue can be solved, we believe it to be equally interesting to investigate other methods for trace-link generation, particularly the use of machine learning. Given enough (correct) datasets that include high- and low-level requirements, as well as correct links, one could use any supervised learning method, for instance Support Vector Machines that have implicit feature selection, to learn a black-box model that can possibly provide better generation of links between requirements. Even more so, one can use ensembles of various supervised learning methods, possibly combined with the VSM algorithm or other information retrieval approaches, to further improve the model.

6 Conclusions

Automatic traceability of requirements is a difficult problem to tackle. In this report, we have explained and implemented 4 different approaches that try to solve this problem with a keyword-based VSM algorithm. Comparing the performance of these approaches, we concluded that, whilst certain methods are performing better than others, the overall performance of these approaches is not sufficient for actual use in real production environment. This means that the problem at hand is not yet solved and, unless a better automatic approach is found, one still has to invest an enormous amount of time into manually maintaining a traceability matrix.

A Appendix

This appendix contains tables that were too large for inserting in the main part of the report. The labels "I+NP" and "NI+P" are constructed using **I:** *identified by experts*, **P:** *predicted by the tool*, and **N:** *not*. As such, "I+NP" refers to *links that are identified by experts AND not predicted by the tool*, whereas "NI+P" refers to *links that are not identified by experts BUT that were predicted by the tool*.

Requirement	I+NP	NI+P
F1		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F2		UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F3		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC15
F4		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC4,UC32,UC15
F5		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F6		UC6,UC19,UC27,UC36,UC24,UC18,UC1,UC16,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC7,UC4,UC15
F7		UC6,UC19,UC27,UC36,UC24,UC18,UC1,UC16,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC7,UC4,UC15
F8		UC6,UC19,UC27,UC36,UC24,UC18,UC1,UC16,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC7,UC4,UC15
F9		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4
F10	UC14	UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F11		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F12		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F13		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F14		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC15
F15		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F16		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F17		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F18		UC6,UC19,UC27,UC36,UC21,UC33,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F19		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F20		UC6,UC19,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F21		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F22		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F23		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F24		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F25		UC6,UC19,UC27,UC36,UC21,UC24,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F26		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F27		UC6,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F28		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F29		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F30		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4
F31		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F32		UC6,UC19,UC27,UC36,UC21,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F33		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC40,UC22,UC7,UC4,UC15
F34		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F35		UC6,UC19,UC27,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F36		UC6,UC19,UC27,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F37		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F38		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F39		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F40		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F41		UC6,UC19,UC27,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F42		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F43		UC6,UC19,UC27,UC36,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F44		UC6,UC19,UC27,UC21,UC33,UC24,UC18,UC1,UC16,UC23,UC31,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC14,UC7,UC4,UC32,UC15
F45		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F46		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F47		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15
F48		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC42,UC35,UC40,UC22,UC7,UC4,UC15
F49		UC6,UC19,UC27,UC36,UC21,UC24,UC18,UC1,UC16,UC23,UC29,UC2,UC25,UC8,UC17,UC35,UC40,UC22,UC7,UC4,UC15

Table 11: Errors for dataset 1 using match type 0

Requirement	I+NP	NI+P
F1	UC1	
F2		UC7,UC23,UC4
F4	UC7	UC4
F5		UC7,UC4
F6	UC22,UC23,UC21	
F7	UC22,UC23,UC21	UC42
F8	UC22,UC23,UC21	UC42
F9	UC15	UC2,UC4
F10	UC14	UC4
F11		UC25,UC2,UC4
F13	UC33	UC16
F14	UC32	UC16
F16	UC31	
F19		UC42
F20		UC2
F23		UC27
F24		UC7,UC2,UC18
F25		UC8,UC2
F27		UC2
F28		UC8,UC18
F29		UC25,UC17
F30		UC17,UC25,UC7,UC2,UC4,UC23,UC16
F31		UC15,UC16
F32	UC24	
F33		UC1
F34		UC35
F36	UC36	
F37		UC24
F42		UC27,UC8,UC2
F44		UC40
F45		UC35
F48		UC15,UC16
F49	UC42	

Table 12: Errors for dataset 1 using match type 1

Requirement	I+NP	NI+P
F2	UC6	UC7
F4	UC7	UC4
F5	UC2	UC4
F6	UC22,UC23,UC21	UC42
F7	UC22,UC23,UC21	UC42
F8	UC22,UC23,UC21	UC42
F9	UC15	UC4
F10	UC14	UC4
F11		UC4
F12		UC16,UC25
F13	UC33	UC7,UC16,UC17
F14	UC32	UC7,UC16,UC17
F16	UC31	UC7
F17		UC27,UC25
F19		UC42
F21		UC25
F22		UC8,UC36,UC7,UC16,UC2
F23		UC27
F25		UC8
F26		UC2
F27		UC2
F28		UC8
F29		UC25,UC17
F31		UC16
F32		UC25
F34		UC35
F37		UC24
F38		UC24
F39		UC7
F40		UC7,UC36
F42		UC8,UC2,UC27
F43		UC18,UC42
F44		UC40
F45		UC35
F46		UC1
F49	UC42	UC40,UC25

Table 13: Errors for dataset 1 using match type 2

Requirement	I+NP	NI+P
F1	UC1	
F2	UC6	UC7
F4	UC7	UC4
F5	UC2	UC4
F6	UC21,UC22,UC23	
F7	UC21,UC22,UC23	UC42
F8	UC21,UC22,UC23	UC42
F9	UC15	UC4
F10	UC14	UC4
F11		UC4
F13	UC33	UC17,UC7,UC16
F14	UC32	UC17,UC7,UC16
F16	UC31	
F19		UC42
F23		UC27
F25		UC8
F26		UC2
F27		UC2
F28		UC8
F29		UC25,UC17
F31		UC16
F32	UC24	
F34		UC35
F36	UC36	
F37		UC24
F42		UC2,UC8,UC27
F44		UC40
F45		UC35
F49	UC42	

Table 14: Errors for dataset 1 using match type 3

Requirement	I+NP	NI+P
F1		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F2		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F3		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F4		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F5		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F6		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F7		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC7
F8		UC12,UC23,UC7,UC9,UC10,UC19,UC20,UC18,UC4,UC3,UC17,UC1,UC5,UC6
F9		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F10		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F11		UC12,UC23,UC21,UC14,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F12		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F13		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F14		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F15		UC12,UC23,UC21,UC14,UC18,UC22,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F16		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F17		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F18		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC3,UC16,UC1,UC5,UC7
F19		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F20		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F21		UC12,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F22		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC16,UC1,UC5,UC7
F23		UC12,UC21,UC14,UC18,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F24		UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F25		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F26		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC16,UC1,UC5,UC7
F27		UC12,UC23,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F28		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F29		UC12,UC23,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F30		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC3,UC16,UC1,UC5,UC7
F31		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F32		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F33		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F34		UC12,UC23,UC21,UC14,UC18,UC22,UC6,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F35		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F36		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F37		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F38		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F39		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F40		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC16,UC1,UC5,UC7

Table 15: Errors for dataset 2 using match type 0 – part 1

Requirement	I+NP	NI+P
F41		UC12,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F42		UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F43		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F44		UC12,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F45		UC12,UC23,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F46		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F47		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F48		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F49		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F50		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F51		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F52		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F53		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F54		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F55		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F56		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F57		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F58		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F59		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC17,UC3,UC16,UC1,UC5,UC7
F60		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F61		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC5,UC7
F62		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F63		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F64		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F65		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F66		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F67		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F68		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F69		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F70	UC7	
F71		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F72		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F73		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F74		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F75		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F76		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5
F77		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC17,UC3,UC16,UC1,UC5,UC7
F78		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC9,UC10,UC19,UC15,UC20,UC11,UC4,UC3,UC16,UC1,UC5,UC7
F79		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC3,UC16,UC1,UC5,UC7
F80		UC12,UC23,UC21,UC14,UC18,UC22,UC13,UC6,UC8,UC10,UC19,UC15,UC20,UC11,UC4,UC3,UC16,UC1,UC5,UC7

Table 16: Errors for dataset 2 using match type 0 – part 2

Requirement	I+NP	NI+P
F2	UC5	
F3	UC5	
F4	UC5	
F5	UC5	
F6	UC5	
F7	UC5	
F8	UC14,UC16,UC15,UC13,UC21,UC11,UC8,UC22	
F9	UC4	
F10	UC4	
F11	UC18	
F12	UC15	
F13		UC13
F14	UC15	
F15	UC13	
F16	UC15	
F17	UC10	
F18	UC17	UC16,UC9
F20	UC19	
F21	UC23	
F22		UC23
F23	UC23	UC3
F24	UC12	
F25	UC15	
F26	UC3	UC20
F27		UC22
F28	UC4	
F30	UC17	
F33	UC7	
F34	UC13,UC8	
F35	UC15	UC16,UC9
F36	UC15	
F39	UC19	
F40	UC3	

Table 17: Errors for dataset 2 using match type 1 – part 1

Requirement	I+NP	NI+P
F42	UC12	UC4
F43	UC15	
F44	UC23	
F46		UC1
F47	UC1	
F48	UC1	
F49	UC1	
F50	UC1	
F51	UC1	
F53	UC1	
F54	UC1	
F55	UC1	
F59	UC4	
F60	UC1	
F61	UC1	
F62	UC15	
F63	UC9	
F64		UC16
F68	UC7	
F69	UC7	
F70	UC7	
F71	UC9	
F72	UC7	
F73	UC7	
F74	UC7	
F75	UC7	
F77		UC16
F78	UC17	UC13,UC16,UC9,UC15
F79	UC17,UC9	
F80	UC17,UC9	

Table 18: Errors for dataset 2 using match type 1 – part 2

Requirement	I+NP	NI+P
F1		UC10,UC16,UC13,UC15,UC20,UC8,UC9,UC4,UC5,UC14,UC7,UC6
F2	UC5	UC10,UC14,UC15
F3		UC19,UC6
F5		UC10,UC16,UC13,UC15,UC20,UC8,UC9,UC4,UC14,UC7,UC6
F6	UC5	UC10,UC11,UC20,UC12,UC17,UC3
F7	UC5	UC10,UC11,UC20,UC12,UC17,UC3
F8	UC16,UC13,UC21,UC22,UC15,UC8,UC14	UC20
F9	UC4	UC1,UC3,UC20
F10	UC4	UC10,UC11,UC20,UC12,UC17,UC3
F11	UC18	UC3,UC20
F12	UC15	UC1,UC3,UC20
F13		UC13
F14		UC10,UC14,UC17,UC12
F15		UC8,UC12
F16		UC10,UC20,UC1,UC12,UC14,UC17,UC3
F17	UC10	UC8,UC13
F18	UC17	UC16,UC9
F20		UC5,UC6
F21	UC23	UC10,UC11,UC20,UC12,UC17,UC3
F23	UC23	UC3
F24	UC12	UC3,UC20
F25	UC15	UC3,UC20
F26	UC3	UC20
F28	UC4	UC10,UC11,UC20,UC12,UC17,UC3
F30	UC17	UC8,UC13,UC12
F33	UC7	UC10,UC15,UC12,UC14,UC17
F35	UC15	UC16,UC9
F36	UC15	UC1,UC3,UC20
F37		UC10,UC14,UC15
F39		UC5,UC6
F40	UC3	UC12
F42	UC12	UC4
F43	UC15	UC1,UC3,UC20
F44	UC23	UC1,UC3,UC20
F46		UC1
F47		UC10,UC13,UC11,UC15,UC20,UC8,UC4,UC12,UC3
F48	UC1	UC10,UC11,UC20
F49		UC8,UC21,UC13
F53	UC1	UC21
F54	UC1	UC10,UC11,UC20,UC12,UC17,UC3
F55	UC1	UC10,UC11,UC20,UC12,UC17,UC3
F59	UC4	UC21
F60	UC1	UC21
F61		UC3,UC20
F62	UC15	UC1,UC3,UC20
F63	UC9	UC1
F64		UC16
F68		UC16,UC9
F69		UC8,UC13
F70		UC23,UC16,UC13,UC11,UC20,UC9,UC12,UC5,UC3,UC17,UC10,UC19,UC15,UC22,UC21,UC8,UC1,UC4,UC18,UC14,UC6
F71	UC9	UC10,UC13,UC11,UC15,UC20,UC8,UC1,UC4,UC12,UC3
F72	UC7	UC10,UC14,UC15
F73	UC7	UC10,UC13,UC11,UC15,UC20,UC8,UC1,UC4,UC12,UC3
F74	UC7	UC10,UC13,UC11,UC15,UC20,UC8,UC1,UC4,UC12,UC3
F75		UC23,UC5,UC19,UC6
F77		UC16
F78	UC17	UC16
F79	UC9,UC17	UC10,UC13,UC11,UC15,UC20,UC8,UC1,UC4,UC12,UC3
F80	UC9,UC17	UC10,UC13,UC11,UC15,UC20,UC8,UC1,UC4,UC12,UC3

Table 19: Errors for dataset 2 using match type 2

Requirement	I+NP	NI+P
F2	UC5	
F3	UC5	
F4	UC5	
F5	UC5	
F6	UC5	
F7	UC5	
F8	UC14,UC16,UC22,UC11,UC8,UC15,UC13,UC21	
F9	UC4	
F10	UC4	
F11	UC18	
F12	UC15	
F13		UC13
F14	UC15	
F15	UC13	
F16	UC15	
F17	UC10	
F18	UC17	UC9,UC16
F20	UC19	
F21	UC23	
F23	UC23	UC3
F24	UC12	
F25	UC15	
F26	UC3	UC20
F28	UC4	
F30	UC17	
F33	UC7	
F34	UC8,UC13	
F35	UC15	UC9,UC16
F36	UC15	
F39	UC19	
F40	UC3	

Table 20: Errors for dataset 2 using match type 3 – part 1

Requirement	I+NP	NI+P
F42	UC12	UC4
F43	UC15	
F44	UC23	
F46		UC1
F47	UC1	
F48	UC1	
F49	UC1	
F50	UC1	
F51	UC1	
F53	UC1	
F54	UC1	
F55	UC1	
F59	UC4	
F60	UC1	
F61	UC1	
F62	UC15	
F63	UC9	
F64		UC16
F68	UC7	
F69	UC7	
F70	UC7	
F71	UC9	
F72	UC7	
F73	UC7	
F74	UC7	
F75	UC7	
F77		UC16
F78	UC17	UC16
F79	UC9,UC17	
F80	UC9,UC17	

Table 21: Errors for dataset 2 using match type 3 – part 2

References

[1] Bird, S. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions* (2006), pp. 69–72.

[2] explosion.ai. spacy. `https://github.com/explosion/spaCy`, 2020.

[3] Hayes, J., Dekhtyar, A., and Osborne, J. Improving requirements tracing via information retrieval. pp. 138– 147.

[4] Lehman, M. M. Programs, life cycles, and laws of software evolution. In *Proceedings of the IEEE, vol. 68, no. 9* (1980), pp. 1060–1076.

[5] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations* (2014), pp. 55–60.