Software Evolution

Assignment 1

23-02-2018

Robin Esposito (1280155) - r.esposito@student.tue.nl
Mattijs Jansen (0748947) - m.j.jansen.1@student.tue.nl

1. Introduction

The goal of this assignment is to implement a tool for trace-link detection, with focus on vertical dependences between high-level and low-level requirements. The tool will take as input two sets of requirements (high-level and low-level) and, after a suitable preprocessing of the text, the requirements will be represented as vectors and compared, in order to evaluate the similarity between them. The output of the detection tool will be a matrix representing, for each high-level requirement, a subset of low-level requirements that are most likely to be linked to said requirement. Traceability has a great importance in the management of requirements evolution, as it allows to identify and support changes in the requirements ([1], Paragraph 1.4.6), but in non-trivial systems the number of requirements results in large amounts of information, which make the detection process too costly to be carried out manually. For this reason there is the need of a trace-link detection tool, which would help software engineer in managing the evolution of software requirements, making their lives easier in the processes of maintenance, testing, certification, reengineering and all the other steps of the process in which the correctness of the requirements is essential.

2. Trace-link detection tool

2.1. Tool development

The tool was implemented using Rascal. This section contains a description of the development steps. In order to analyze the content of the requirements, the first step consisted in preprocessing the data: the high-level and low-level requirement were read from the respective files and saved as two different lists, in which each element corresponds to a requirements, shaped as a list of words. The words were then converted to lowercase, to avoid string matching errors, and filtered by removing the stop words. The list of stop words for the english language was obtained from the XPO6 website [2].

The stemming process was implemented by using the *"stem()"* function, included in Rascal in the *analysis::stemming::Snowball* module. Stemming consists in reducing every word to a basic form, in order to ignore the differences in the endings of the terms caused by plurals or verb conjugation. Once all terms have been stemmed, adapting the stemmer algorithm to the English language, they can be used to evaluate the similarity of semantic with a better accuracy. The words obtained so far, contained in the two requirements list, were used to create a single master vocabulary, containing all the terms used in the description of the requirements: this was implemented in the tool by using a set of strings, in order to automatically exclude all duplicates.

The following steps consisted in creating a vectorial representation of each requirement, using the *tf*ifd* score which will later be used to compute similarity ([3], Section 3). This was done by using two nested “for” loops, repeating the process twice for high-level requirements and low-level requirements. The outer loop iterates over the list of requirements, while the inner loop over the words in the vocabulary. For every word the tools checks if that word is contained in the requirement: if it is not present, than a value “0” is added to the vector, otherwise the *tf*ifd* score is computed. *Tf* indicates how important a term in relation to the full requirement text, measuring how frequently the word appears in the current requirement, while *idf* is an indication of the significance of the word in relation to the full collection of documents, and is obtained as $log_2(n/d)$, where *n* is the total number of requirements and *d* is the number of requirements in which that specific term appears.

The vectorial representation of the requirements was then used to build the similarity matrix, in which rows are high-level requirements and columns are low-level requirements, and each cell contains the value of the cosine similarity between the two corresponding vectors. These values were obtained by implementing the following formula [3]:

$$sim(x,y) = \frac{\sum_{i=1}^{n} x_i * y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}$$

Lastly, using the similarity matrix described above, it was possible to implement the actual detection of the trace-links. According to the requirements, the user can run the detection tool using five different filtering modes, which result in five different output files. The first mode consists in a null filtering, therefore all the low-level requirements are associated to each high-level requirement, provided that the similarity value is non-negative. If the second mode is chosen, the tools reports the four low-level requirements with highest similarity. This is achieved by creating a dictionary, implemented as a map object with an element for each low-level requirement, in which the name of the requirement is the key and the similarity score is the value. Using this dictionary, the maximum value is retrieved and the process is iterated four times: in order to avoid duplicates the current maximum is removed from the dictionary at the end of each cycle. The name of the four requirements with the highest scores is then printed in the corresponding line in the output file. This process would be more immediate if it was possible to order the map elements using their values and then extracting the first four elements, but since such functionalities were not found in the Rascal documentation, the described implementation was considered to be the best possible option.

The third method is implemented in a similar way to the first one, but in this case the requirements are selected only if the similarity score is at least *0.25*. For methods 4 and 5, the first operation is to find the highest similarity value in the vector. Then, this value is used as a threshold to filter the low-level requirements: a trace-link is detected if its similarity value is at least *0.67* or *0.50* times the threshold. The detected trace-links are finally printed in a text file, which is formatted similarly to the MODIS hand trace file, with each line containing an high-level requirements and its respective links.

2.2. Running the tool

The toon can be run in two modes: detection and evaluation. The implementation of the evaluation will be discussed in Section 4: here we just describe how to run it. After starting the Rascal console and importing the "Main" module (using "*import Main;*"), it is possible to invoke the two respective methods using the following signatures:

```
detect(|project://Assignment1/data/modis/high|, |project://Assignment1/data/modis/low|, 1);

evaluate(|project://Assignment1/data/modis/high|, |project://Assignment1/data/modis/low|, 1,
                    |project://Assignment1/data/modis/handtrace.txt|);
```

The first three parameters are the same in both cases: they represent the directories of high and low requirements and the type of filtering that the user wishes to apply (an integer between 1 and 5). The fourth parameter in the evaluate method represents the location of the manually generated handtrace file, provided by the MODIS dataset. All paths have to be surrounded by the character "|", as the tool expects objects of type *loc*. To ease the running of the tool two more functions were implemented: *runTool()* and *evaluateTool()*. These methods simply encapsulate the two main functionalities and invoke the respective methods, using the default parameters described above.

3. MODIS dataset

The MODIS dataset contains 19 high-level requirements and 49 low-level requirements. It also provides a *"handtrace.txt"* file, in which are represented the manually obtained link traces that will be used in Section 4 to evaluate the tool. This file represents the links between high-level and low-level requirement using the following format: every line is associated to a high-level requirement, the lines are separated by "%" and each of them contains the name of an high-level requirement, followed by the names of the associated low-level requirements.

4. Evaluation

In this section the five different filtering methods are evaluated. For each technique a confusion matrix is made, which displays the true positives, false positives, false negatives and true negatives for the detection tool. Furthermore, the recall score is determined based on the percentage of true positives compared to all manually identified trace-links and precision score is calculated as the percentage true positives compared to all the trace-links found by the detection tool.

4.1

The first case is the no filtering technique, the similarity score between high-level requirements and low-level requirement is at least 0. The confusion matrix in Table 1 shows the results of this technique. From the confusion matrix it can be calculated that the recall is 100% and the precision is 4.4%. Because the similarity score between a high-level requirement and a low-level requirement has to be at least 0, the tool predicts a trace-link between all the high-level requirements and low-level requirements, which are all possible trace-links. Therefore the misclassifications of this technique are not listed.

Table 1: Confusion matrix for technique 1 to predict trace-links.

	trace-link identified manually	trace-link not-identified manually
trace-link predicted by the tool	41	890
trace-link not predicted by the tool	0	0

4.2

The second method that is evaluated are the top 4 low-level requirements with the highest similarity score for each high-level requirement. In Table 2 the results of this technique can be found. This results in a recall of 31.7% and a precision of 17.1%.

Table 2: Confusion matrix for technique 2 to predict trace-links.

	trace-link identified manually	trace-link not-identified manually
trace-link predicted by the tool	13	63
trace-link not predicted by the tool	28	827

The false negative and false positive misclassifications are listed respectively in Table A1 and Table A2 in Appendix A.1. For each high-level requirement four traces are detected by the tool. Because for most high-level requirements only one, two, or three traces are found manually and for some none, this explains that the tool predicts many links that are not identified manually. For the requirement SDP5.2-1 22 traces are detected manually. If only the top 4 traces are predicted by the tool, already 18 traces will not be found by the tool, which explains the low recall value.

4.3

For the third technique the similarity score has to be 0.25 or above. The results are displayed in Table 3 and following from the confusion matrix the recall is 14.6% and the precision is 75.0%. The mistakes made by the tool can be found in Table A3 and A4 in the Appendix A.2. Only two false positives are detected, which are caused by the linked requirements having multiple words which are the same for both requirements. The offset of 0.25 results in only six well-predicted trace-links. For the other 35 manually found traces this offset is too high, no or only one or two words are the same for both the high-level and low-level requirement.

Table 3: Confusion matrix for technique 3 to predict trace-links.

	trace-link identified manually	trace-link not-identified manually
trace-link predicted by the tool	6	2
trace-link not predicted by the tool	35	888

4.4

The filtering of the fourth technique consist of reporting the trace-links that have at least a similarity value of 67% of the highest value in the similarity matrix. The results are shown in Table 4. From the table the recall value of 26.8% and the precision value of 26.8% are retrieved.

Table 4: Confusion matrix for technique 4 to predict trace-links.

	trace-link identified manually	trace-link not-identified manually
trace-link predicted by the tool	11	30
trace-link not predicted by the tool	30	860

The miscalculations for this technique can be found in Tables A5 and A6 in Appendix A.3. The offset for this technique is lower than for the third technique. This results in higher true positives and lower false negatives predicted trace-links, however, there are more false positives predicted trace-links.

4.5

The last technique that is evaluated reports the trace-links that have at least a similarity value of 50% of the highest value in the similarity matrix. The results for this evaluation are displayed in Table 5 and the recall and precision are respectively 34.1% and 19.7%. The mistakes for this technique can be found in Table A7 and Table A8 in Appendix A.4. The tool will detect a trace-link for a lower similarity score than for techniques 3 and 4. The result is that compared to technique 3 three more trace-links are predicted correct. However, 27 more trace-links are detected spuriously.

Table 5: Confusion matrix for technique 5 to predict trace-links.

	trace-link identified manually	trace-link not-identified manually
trace-link predicted by the tool	14	57
trace-link not predicted by the tool	27	833

4.6

In this part the five different techniques are compared. The recall and precision scores of these five techniques are displayed in Table 6. No filtering has the highest recall, however all possible trace-links are detected and therefore the precision is the lowest of all techniques and therefore this technique is not helpful to predict trace-links well. The other four techniques all have trade-offs. In Table 6 can be seen that technique 5 has both a higher recall and precision percentage than technique 2. However, only one dataset is used to evaluate the different methods, from which it can no be concluded that technique 5 is better than technique 2 for other datasets.

Techniques 3, 4 and 5 have different offsets for the similarity score and therefore many of the mistakes made by the tool are the same for these filtering techniques. Technique 2 uses another approach for the detection of trace-links. The result is that most of the false positives are different. The false negatives are however mostly the same.

Table 6: Recall and precision for each of the five techniques.

	no filtering	top 4	0.25	0.67 max_sim	0.5 max_sim
recall	100%	31.7%	14.6%	26.8%	34.1%
precision	4.4%	17.1%	75.0%	26.8%	19.7%

5. Discussion

One possible way to improve the trace-link detection tool is to normalize the values of words to the number of words a requirement consists of. Matching keywords of shorter sentences will result in a higher similarity score compared to longer sentences with the same number of matching keywords.
Another option to improve the tool could be to implement the vectorial representation in a different way. Instead of using the *tf*ifd* score, vectors could be encoded using a context based approach, such as the Word2vec skip-gram model. In this way we would obtain a vectorial representation of each word in the text of requirements, and those value could be used to compute a more accurate score of the similarity between a low-level and a high-level requirement. This variant would naturally be slower, but could improve the detection results.

6. Conclusion

In this report we have shown an implementation of a tool to detect traceability links between high-level and low-level requirements. Five different filtering techniques were used to evaluate the detection tool for the MODIS dataset. All different methods have trade-offs, however, techniques 3 and 5 seem to have the best result depending on if the user of the tool wants a under- or over-approximation. Changing the offset to a lower or higher value will increase respectively the over-approximation or under-approximation.

7. References

[1] Tom Mens, Alexander Serebrenik, Anthony Cleve (eds.) Evolving Software Systems, 2014.

[2] List of English Stop Words, http://xpo6.com/list-of-english-stop-words/, 2009.

[3] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne, Improving Requirements Tracing via Information Retrieval, September 2003, Monterey Bay, CA.

Appendix A

A.1

Table A1: False negative misclassifications for case 2

SDP3.2-2	L1APR01-I-3
SDP3.3-4	L1APR03-I-2 L1APR01-I-2
SDP5.2-1	L1APR01-F-1.1-5 L1APR01-F-1.2-5 L1APR01-F-2.1-4 L1APR01-F-2.1-5 L1APR01-F-2.2.2-4 L1APR01-F-2.2.3-4 L1APR01-F-2.2.4-4 L1APR01-F-2.3-5 L1APR01-F-4-5 L1APR03-F-2.4-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2 L1APR03-F-3.2.3-2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR02-F-4.4-2 L1APR03-F-5.4-2 L1APR03-F-5.5-2
SDP5.2-4.3	L1APR01-F-2.2.4-2
SDP5.2-4.5	L1APR01-F-1.2-1
SDP5.3-1	L1APR03-F-2.5-5 L1APR03-F-1-2 L1APR03-F-2.5-1

Table A2: False positive misclassifications for case 2

SDP3.2-2	L1APR03-F-5.4-2 L1APR01-I-1 L1APR03-F-2.4-1 L1APR01-F-4-4
SDP3.3-1	L1APR03-F-1-2 L1APR01-F-2.2.3-4 L1APR01-F-2.1-1
SDP3.3-2	L1APR03-F-1-2 L1APR01-F-2.2.4-1
SDP3.3-4	L1APR01-F-2.2.3-4 L1APR03-F-1-2 L1APR01-F-2.1-1
SDP4.2-1	L1APR01-I-1 L1APR01-F-2.4-2 L1APR01-F-2.4-1
SDP4.2-2	L1APR01-F-4-5 L1APR01-F-4-4 L1APR01-I-1
SDP4.2-3	L1APR01-F-2.1-1 L1APR01-F-4-5 L1APR01-F-4-4
SDP5.2-1	L1APR01-I-2 L1APR03-I-2
SDP5.2-3	L1APR03-F-2.5-2 L1APR03-F-5.5-2 L1APR03-F-2.4-2
SDP5.2-4.3	L1APR03-F-2.5-2 L1APR03-F-2.4-2 L1APR03-F-2.5-5 L1APR01-F-1.2-1
SDP5.2-4.5	L1APR03-F-2.5-2 L1APR03-F-1-2

SDP5.3-1	L1A5.2 L1APR02-F-4.1-2 L1APR03-F-4.2-2
SDP4.3-1	L1APR01-F-4-5 L1APR01-F-4-4 L1APR01-I-2 L1APR03-I-2
SDP4.3-2	L1APR03-F-3.2.3-2 L1APR03-F-3.4.4-1 L1APR03-I-5 L1APR01-F-1.1-1
SDP6.1-1	L1APR01-I-1 L1APR01-F-2.2.3-4 L1APR01-F-2.1-4 L1APR01-F-2.2.4-2
SDP6.1-2	L1APR03-F-5.4-2 L1APR01-F-2.2.3-4 L1APR01-I-1 L1APR01-F-2.1-4
SDP6.1-3	L1APR01-I-1 L1APR01-F-4-3 L1APR03-F-3.4.4-1 L1APR01-F-2.2.3-4
SDP6.1-4	L1APR01-I-1 L1APR03-F-2.5-2 L1APR01-F-2.2.3-4 L1APR03-F-3.2.1-2
SDP6.1-5	L1A5.2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR02-F-4.4-2
SDP6.1-5	L1A5.2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR02-F-4.4-2

A.2

Table A3: False negative misclassifications for case 3

SDP3.2-2	L1APR01-I-3
SDP3.3-1	L1APR01-I-1
SDP3.3-2	L1APR03-I-2 L1APR01-I-2
SDP3.3-4	L1APR03-I-2 L1APR01-I-2
SDP5.2-1	L1APR01-F-1.1-5 L1APR01-F-1.2-5 L1APR01-F-2.1-4 L1APR01-F-2.1-5 L1APR01-F-2.2.2-4 L1APR01-F-2.2.3-4 L1APR01-F-2.2.4-4 L1APR01-F-2.3-5 L1APR01-F-4-5 L1APR03-F-2.4-1 L1APR03-F-2.4-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2 L1APR03-F-3.2.3-2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR02-F-4.4-2 L1APR03-F-5.4-2 L1APR03-F-5.5-2
SDP5.2-4.3	L1APR01-F-2.2.4-2
SDP5.2-4.5	L1APR01-F-1.2-1 L1APR03-F-5.5-2 L1APR03-F-5.4-2
SDP5.3-1	L1APR03-F-2.5-5 L1APR03-F-1-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2

Table A4: False positive misclassifications for case 3

SDP5.2-4.3	L1APR03-F-2.5-2
SDP6.1-3	L1APR01-I-1

A.3

Table A5: False negative misclassifications for case 4

SDP3.2-2	L1APR01-I-3
SDP3.3-4	L1APR03-I-2 L1APR01-I-2
SDP5.2-1	L1APR01-F-1.1-5 L1APR01-F-1.2-5 L1APR01-F-2.1-4 L1APR01-F-2.1-5 L1APR01-F-2.2.2-4 L1APR01-F-2.2.3-4 L1APR01-F-2.2.4-4 L1APR01-F-2.3-5 L1APR01-F-4-5 L1APR03-F-2.4-1 L1APR03-F-2.4-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2 L1APR03-F-3.2.3-2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR02-F-4.4-2 L1APR03-F-5.4-2 L1APR03-F-5.5-2
SDP5.2-4.3	L1APR01-F-2.2.4-2
SDP5.2-4.5	L1APR01-F-1.2-1
SDP5.3-1	L1APR03-F-2.5-5 L1APR03-F-1-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2

Table A6: False positive misclassifications for case 4

SDP3.2-2	L1APR03-F-5.4-2
SDP3.3-1	L1APR03-F-1-2
SDP3.3-2	L1APR03-F-1-2
SDP4.2-3	L1APR01-F-2.1-1 L1APR01-F-4-5
SDP5.2-4.3	L1APR03-F-2.5-2
SDP5.2-4.5	L1APR03-F-1-2 L1APR03-F-2.4-2 L1APR03-F-2.5-2 L1APR03-I-5
SDP5.3-1	L1A5.2
SDP4.3-1	L1APR01-F-4-4 L1APR01-F-4-5 L1APR01-I-2 L1APR03-I-2
SDP4.3-2	L1APR03-F-3.2.3-2

SDP6.1-1	L1APR01-F-2.2.3-4 L1APR01-I-1
SDP6.1-2	L1APR03-F-5.4-2
SDP6.1-3	L1APR01-I-1
SDP6.1-4	L1APR01-F-2.2.3-4 L1APR01-I-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2
SDP6.1-5	L1A5.2 L1APR02-F-4.1-2 L1APR02-F-4.4-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR03-I-5

A.4

Table A7: False negative misclassifications for case 5.

SDP3.2-2	L1APR01-I-3
SDP3.3-4	L1APR03-I-2 L1APR01-I-2
SDP5.2-1	L1APR01-F-1.1-5 L1APR01-F-1.2-5 L1APR01-F-2.1-4 L1APR01-F-2.1-5 L1APR01-F-2.2.2-4 L1APR01-F-2.2.3-4 L1APR01-F-2.2.4-4 L1APR01-F-2.3-5 L1APR01-F-4-5 L1APR03-F-2.4-1 L1APR03-F-2.4-2 L1APR03-F-2.5-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2 L1APR03-F-3.2.3-2 L1APR02-F-4.1-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR02-F-4.4-2 L1APR03-F-5.4-2 L1APR03-F-5.5-2
SDP5.2-4.3	L1APR01-F-2.2.4-2
SDP5.2-4.5	L1APR01-F-1.2-1
SDP5.3-1	L1APR03-F-2.5-5

Table A8: False positive misclassifications for case 5.

SDP3.2-2	L1APR01-I-1 L1APR03-F-5.4-2
SDP3.3-1	L1APR01-F-2.2.3-4 L1APR03-F-1-2
SDP3.3-2	L1APR01-F-2.2.4-1 L1APR03-F-1-2
SDP4.2-2	L1APR01-F-4-5
SDP4.2-3	L1APR01-F-1.1-1 L1APR01-F-2.1-1 L1APR01-F-4-4 L1APR01-F-4-5

SDP5.2-1	L1APR01-I-2 L1APR03-I-2
SDP5.2-4.3	L1APR01-F-1.2-1 L1APR03-F-2.4-2 L1APR03-F-2.5-2 L1APR03-F-2.5-5
SDP5.2-4.5	L1APR01-F-1.1-1 L1APR01-F-2.1-1 L1APR01-I-1 L1APR03-F-1-2 L1APR03-F-2.4-2 L1APR03-F-2.5-2 L1APR03-I-5
SDP5.3-1	L1A5.2 L1APR01-F-4-3 L1APR02-F-4.1-2 L1APR02-F-4.4-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2
SDP4.3-1	L1APR01-F-2.2.4-1 L1APR01-F-4-4 L1APR01-F-4-5 L1APR01-I-2 L1APR03-I-2
SDP4.3-2	L1APR03-F-3.2.3-2
SDP6.1-1	L1APR01-F-2.1-4 L1APR01-F-2.2.3-4 L1APR01-I-1
SDP6.1-2	L1APR03-F-5.4-2
SDP6.1-3	L1APR01-I-1
SDP6.1-4	L1APR01-F-2.1-1 L1APR01-F-2.1-4 L1APR01-F-2.2.3-4 L1APR01-F-4-3 L1APR01-I-1 L1APR03-F-2.5-2 L1APR03-F-3.2.1-2 L1APR03-F-3.2.3-2
SDP6.1-5	L1A5.2 L1APR01-F-4-5 L1APR02-F-4.1-2 L1APR02-F-4.4-2 L1APR03-F-2.5-2 L1APR03-F-4.2-2 L1APR03-F-4.3-2 L1APR03-I-5