# Mining Requirements Links

Vincenzo Gervasi[1,2] and Didar Zowghi[2]

[1] Dipartimento di Informatica, University of Pisa, Italy
[2] School of Software, University of Technology, Sydney, Australia

**Abstract. [Context & motivation]** Obtaining traceability among requirements and between requirements and other artifacts is an extremely important activity in practice, an interesting area for theoretical study, and a major hurdle in common industrial experience. Substantial effort is spent on establishing and updating such links in any large project – even more so when requirements refer to a product family. **[Question/problem]** While most research is concerned with ways to reduce the effort needed to establish and maintain traceability links, a different question can also be asked: how is it possible to harness the vast amount of implicit (and tacit) knowledge embedded in already-established links? Is there something to be learned about a specific problem or domain, or about the humans who establish traces, by studying such traces?

**[Principal ideas/results]** In this paper, we present preliminary results from a study applying different machine learning techniques to an industrial case study, and test to what degree common hypothesis hold in our case. **[Contribution]** Reshaping traceability data into knowledge can contribute to more effective automatic tools to suggest candidates for linking, to inform improvements in writing style, and at the same time provide some insight into both the domain of interest and the actual implementation techniques.

**Keywords:** traceability, machine learning, knowledge mining.

## 1 Introduction

A famous definition of *Traceability* in requirements is provided in [4]:

> "Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)."

This definition places particular emphasis on the ability to *follow* the life of requirements, or in other words, on following the *links* established between requirements and other artifacts, or among requirements (which, in turn, can include requirements at different stages of evolution, or business requirements to requirements specifications, or various requirements in a single requirements document which taken together describe a single feature, etc.).

The major hurdle to ensure traceability is the effort needed to establish and maintain the links between all those artifacts, while the artifacts themselves undergo their evolution. It is not surprising then that most research in the area has aimed at facilitating the establishment of links, typically by providing semi-automatic tools to that end. In the most common approach, Information Retrieval techniques such as the Vector Space Model (VSM) or Latent Semantic Indexing (LSI), are used to identify a set of candidate requirements to be linked, based on the similarity of terms they contain [2,6,5].

In this paper we take the dual approach: instead of asking ourselves how to suggest traceability links, we investigate what can be learned from links that are already established. The situation where a set of links is already established is in fact common in industrial practice, especially in large or long-lived projects. In that context, other factors compound the problem: for example, changing teams means that links established at different times follow different conventions for what is relevant; requirements databases being exchanged between organizations (e.g., from a contractor to the main company, or when actual programming is outsourced outside of the main company) means that a full set of requirements and links, established according to unfamiliar or inconsistent principles, are acquired together.

## 2 Case Study and Experiments

We used a publicly-available dataset of requirements with traceability information, originally based on the CM-1 project by the NASA Metrics Data Program [1]. The dataset comprises 235 high-level SRS (software requirements specification) which are refined to 220 low-level SDS (software design specification) for the same DPU (data processing unit); 361 manually-verified links relate the two sets and, so to say, "tell the story" of the refinement. Notice the 361 links constitute just the 0.7% of all possible pairwise[1] links between SRS and SDS, so the linking relationship in the dataset is very selective. On this dataset we ran two series of experiment, described in the following.

### 2.1 Using Machine Learning to Infer Traces

Most approaches to semi-automatic tracing are based on the assumption that the occurrence of similar terms in two requirements increases the likelihood of them being related by a link (with [3] and [7] being recent exceptions). This likelihood is estimated with techniques such as VSM based on *tf-idf* (term frequency-inverse document frequency), where very common terms contribute less to the similarity score than highly-specialized ones that only appear in few places. Normally, this hypothesis is tested by verifying how many suggested links above a certain likelihood threshold are correct, thus giving rise to the two usual metrics of *precision* (which percentage of the suggested links are correct) and *recall* (which percentage of the correct links were suggested).

We set instead to verify if the hypothesis itself (i.e., that the occurrence of the same terms in two requirements implies greater link affinity) could be mined from the available set of correct links. This was obtained through the following procedure:

[1] In theory traceability should be $n$:$m$, i.e. a set of elements to another set of elements, but in practice most industrial tools have them as 1:1 relations, and their possible grouping is left to the interpretation of the reader. We will submit to the common usage here.

1. All requirements were tokenized[2] and stemmed; stopwords where then removed, thus obtaining a set of 1785 terms that constitute the domain vocabulary employed in our requirements (only 268 of these appeared in both SRS and SDS).
2. From each term $t$ in the vocabulary, two features were derived, one for the occurrence of $t$ in a high-level requirement (named $t_H$) and one for the similar occurrence in a low-level requirement (named $t_L$), giving in total 3570 features.
3. Each requirement was then transformed into a vector of features, with each feature having the tf-idf value of the corresponding term. Higher values indicate higher significance, with 0 indicating a non-occurring term.
4. From these vectors was derived a set of *classification cases* by joining one high-level requirement and one low-level requirement, and adding a classification of `link` or `nolink` based on whether that particular pair was a true link in the original dataset, or not. To facilitate application of the machine learning algorithms, the set was composed of 722 classification cases, half each for `link` and `nolink`. Notice that this alters the statistical features of the set (we have 50% links compared to 0.7% in the original dataset); this issue will be discussed later.
5. Finally, the dataset was used to train and test two different classifiers from the WEKA [8] collection, a Naive Bayesian classifier and the J48 decision-tree classifier (based on the C4.5 algorithm), in a standard 10-fold cross-validation scheme, and both the structure of the classifiers obtained, and the evaluation of their performances, were analyzed.

## 2.2 Using Traces to Infer Domain Thesaurus

In the second experiment, we considered whether the existing traces could suggest stronger affinity between different terms, and weaker affinity between the same terms, compared to the basic hypothesis of VSM that affinity coincides with identity (i.e., only the occurrence of the same term contributes to estimating the probability of a link).

To this end, we derived an affinity score $a$ for each pair of terms $(p_H, q_L)$ as follows:

$$a(p_H, q_L) = \sum_{P,Q} s(P,Q) \cdot (t(p_H, P) + t(q_L, Q))$$

where $P$ is a high-level requirement, $Q$ is a low-level requirement, $s(P,Q)$ is $+1$ if $P$ and $Q$ are linked or $-1$ otherwise, and $t(r,R)$ is the tf-idf value for term $r$ in requirement $R$. For the purpose of this particular experiment, we do not concern ourselves with scaling the values based on document size, since we are considering a single dataset.

With the formula above, terms that occur particularly often in linked requirements pairs, and not commonly in unlinked requirements, would have positive affinity; neutral terms would have an affinity close to 0; negative affinity indicates that those terms tend to appear more frequently in unlinked requirements.

The null hypothesis is that pairs $(t_H, t_L)$ should have high affinity, meaning that the occurrence of the same term in a high-level and a low-level requirement is an indication

[2] A purely alphabetic tokenizer was used; this simplistic choice caused the breaking up of acronyms such as "DPU-1553", "DPU-BOOT", "BIT_DRAM" which could be considered a single term. On the other hand, cases such as "write/read/compare" were correctly split.

that they should be linked. In contrast, pairs $(p_H, q_L)$ with high affinity, where $p \neq q$, indicate that $p$ and $q$ are strictly related terms, despite being different.

## 3 Results and Discussion

The Naive Bayes classifier obtained 60.5% precision and 60.1% recall, whereas the J48 one obtained 67.6% for both, which in the given conditions is a 17.6% improvement over the 50% a trivial classifier would obtain (e.g., one that classifies all pairs as `link`, or as `nolink`, or at random). This figure should be compared with the one obtained by traditional VSM, which for our case is 86.5% at a threshold of 0.04 (see Figure 1). Notice again that these figures are somewhat artificial, in that in common practice, the number of non-links is often a hundredfold greater than the number of links, whereas in our sample they were forcibly set equal[3].

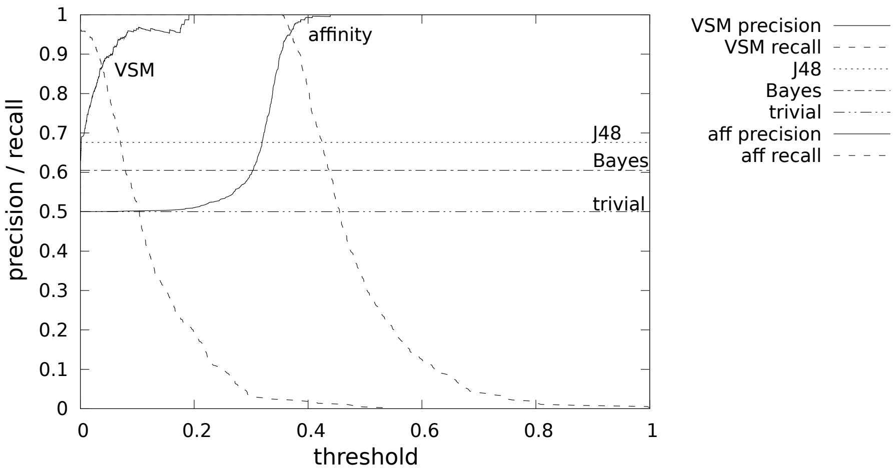

**Fig. 1.** Compared performances of VSM, J48, Naive Bayesian, trivial and affinity classifiers on predicting links for the CM-1 case study

As Figure 1 shows, VSM is a clear winner over both Naive Bayesian and J48. However, the 17.6% improvement the latter has over the trivial classifier was obtained *without* recourse to the VSM hypothesis that occurrence of the same terms in two requirements imply greater link affinity. In fact, direct inspection of the decision tree learned by the J48 classifier shows very few instances in which the appearance of the same term in both high-level and low-level requirements is a crucial factor for the classification decision. This observation suggests that further improvements over VSM can be obtained by harnessing the models mined by training the classifiers.

The good performance of VSM can also be described via the affinity scores of identical pairs, e.g. $a(t_H, t_L)$. Indeed, based on the results of our second experiment

[3] If we had used the natural distribution, a trivial classifier always answering `nolink` would have scored 99.3%, making any relative improvement very difficult to measure.

**Table 1.** Distribution of affinity for identical pairs and for all pairs

| $a(p_H, q_L)$ | $> 0$ | $= 0$ | $< 0$ | N |
|---|---|---|---|---|
| identical pairs | 89% | 4% | 7% | 268 |
| all pairs | 45% | 6% | 48% | 134531 |

**Table 2.** Best and worst affinity scores for identical and non-identical pairs of terms

| $a(t_H, t_L)$ | | | $a(p_H, q_L)$ with $p \neq q$ | | |
|---|---|---|---|---|---|
| 90.620 | bit$_H$ | bit$_L$ | 124.779 | boot$_H$ | dram$_L$ |
| 84.640 | ssi$_H$ | ssi$_L$ | 114.231 | boot$_H$ | bit$_L$ |
| 77.838 | ccm$_H$ | ccm$_L$ | 113.881 | boot$_H$ | bootstrap$_L$ |
| 73.437 | dram$_H$ | dram$_L$ | 112.550 | csc$_H$ | dram$_L$ |
| 71.521 | icu$_H$ | icu$_L$ | 97.982 | boot$_H$ | eeprom$_L$ |
| 59.682 | error$_H$ | error$_L$ | 97.419 | boot$_H$ | test$_L$ |
| ... | | | ... | | |
| -6.962 | allocate$_H$ | allocate$_L$ | -54.900 | dpu$_H$ | data$_L$ |
| -7.360 | software$_H$ | software$_L$ | -57.036 | dpu$_H$ | dpa$_L$ |
| -28.600 | csc$_H$ | csc$_L$ | -60.287 | boot$_H$ | data$_L$ |

(see Table 1), almost 90% of the terms had a positive affinity with themselves. It is thus clear that VSM, where it is assumed that all terms have positive affinity with themselves, provides a good approximation of the real traces.

Still, better can be done by mining the affinity scores from trace data, and using those to refine the results of VSM. In fact, not only 11% of the terms did not satisfy VSM's underlying assumption (and their use in predicting traces was thus detrimental), but a large number of non-identical pairs had even higher affinity scores than the identical pairs. For example, Table 2 lists the highest and lowest affinity scores in our case. On one side, it is interesting to note how most terms shown are domain-specific acronyms; these tend to characterize most strongly the subject of a requirement. In contrast, most common terms (e.g., state, command, application, routine, etc.) tend to have neutral affinity scores (i.e., closer to 0). On the other side, it can be observed how the affinity of certain non-identical pairs (e.g., boot$_H$ with DRAM$_L$, bit$_L$, bootstrap$_L$, EEPROM$_L$) is higher than that of identical pairs. It turns out that using affinity scores instead of cosine similarity on the tf-idf vectors, in our case we have a remarkable maximum of 95.7% simultaneous precision and recall, vs. 86.5% for VSM (Figure 1).

Another interesting observation is that from affinity data we can obtain some insight into the domain. For example, CSC$_H$ and CSC$_L$ have negative affinity (Table 2); investigation in the domain reveals that "CSC" is a generic term from "(sub)system", hence it is not surprising that different occurences of "CSC" refer to different things, and do not imply a relationship between requirements. On the contrary, boot$_H$ and bootstrap$_L$ have high affinity, for obvious linguistic reasons that would be missed by VSM. More interestingly, boot$_H$ and dram$_L$also have high affinity: something that is explained only when realizing that, in the implementation, the DRAM of the system is initialized from stored values at boot time.

## 4 Conclusions

In this preliminary study, we have identified some of the information that can be mined from requirements traces, showing that "there is life beyond VSM". In our case study, we harnessed two such sources of information: (i) the decision tree generated by the J48 machine-learning algorithm, and (ii) the affinity measure we defined above. In both cases, the additional knowledge gained could be used to help familiarize with an unknown domain, to shed some light on refinement decisions, to understand linking policies, or – in the end – to obtain a more accurate semi-automatic linking of new or changed requirements based on previous history.

In further pursuing this line of research, we will investigate how the various techniques we employed behave on data with a more realistic distribution (namely, with less than 1% of all possible pairs of requirements linked instead of 50% as in this preview), a study which is rendered difficult by the substantial computational power needed to process large datasets. We will also test how to best integrate the affinity measures we mined from the data, in order to improve the results from well-established techniques.

We will also extend the analysis to more datasets, to assess how generalizable the finding from the CM-1 case study are, and explore how data mined from existing links can be best visualized and exploited for other tasks, e.g. to infer a domain glossary.

**Acknowledgements.** The authors would like to thank Jane Hayes for her help in accessing the data and inspiration with this work. This material is based upon work supported by the National Science Foundation under Grant No. 0811140.

## References

1. Center of excellence for software traceability, `http://www.traceabilitycenter.org/`
2. Cleland-Huang, J., Berenbach, B., Clark, S., Settimi, R., Romanova, E.: Best practices of automated traceability. IEEE Computer 40(6) (June 2007)
3. Cleland-Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A machine learning approach for tracing regulatory codes to product specific requirements. In: Proc. of the 32nd Int. Conf. on Software Engineering, pp. 155–164. ACM, New York (2010)
4. Gotel, O., Finkelstein, A.: An analysis of the requirements traceability problem. In: Proc. of the First Int. Conf. on Requirements Engineering, pp. 94–101. IEEE CS Press, Los Alamitos (1994)
5. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proc. of the 25th Int. Conf. on Software Engineering, pp. 125–137 (2003)
6. Nattoch Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B.: Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In: Proc. of the 12th IEEE Int. Requirements Engineering Conf., IEEE CS Press, Los Alamitos (2004)
7. Sultanov, H., Hayes, J.H.: Application of swarm techniques to requirements engineering: Requirements tracing. In: Proc. of the 18th IEEE Int. Requirements Engineering Conf., pp. 211–220. IEEE CS Press, Los Alamitos (2010)
8. Weka 3: Data mining software in Java, `http://www.cs.waikato.ac.nz/ml/weka/`