Tracing Requirements in Software Design

Zeheng Li
Southern Methodist University
Dallas, TX 75275-0122, USA
zehengl@smu.edu

Mingrui Chen
Southern Methodist University
Dallas, TX 75275-0122, USA
mingruic@smu.edu

LiGuo Huang
Southern Methodist University
Dallas, TX 75275-0122, USA
lghuang@smu.edu

Vincent Ng
University of Texas at Dallas
Richardson, TX 75083-0688, USA
vince@hlt.utdallas.edu

Ruili Geng
Spectral MD
Dallas, TX 75201, USA
rgeng@smu.edu

ABSTRACT

Software requirement analysis is an essential step in software development process, which defines what is to be built in a project. Requirements are mostly written in text and will later evolve to fine-grained and actionable artifacts with details about system configurations, technology stacks, etc. Tracing the evolution of requirements enables stakeholders to determine the origin of each requirement and understand how well the software's design reflects to its requirements. Reckoning requirements traceability is not a trivial task, we focus on applying machine learning approach to classify traceability between various associated requirements. In particular, we investigate a *2-learner*, *ontology-based* approach, where we train two classifiers to separately exploit two types of features, *lexical* features and features derived from a hand-built ontology. In comparison to a supervised baseline system that uses only lexical features, our approach yields a relative error reduction of 25.9%. Most interestingly, results do not deteriorate when the hand-built ontology is replaced with its automatically constructed counterpart.

CCS CONCEPTS

• **Software and its engineering → Requirements analysis**; *Software design engineering*;

KEYWORDS

Requirements Traceability, Software Design, Machine Learning

ACM Reference format:
Zeheng Li, Mingrui Chen, LiGuo Huang, Vincent Ng, and Ruili Geng. 2017. Tracing Requirements in Software Design. In *Proceedings of 2017 International Conference on Software and Systems Process, Paris, France, July 5–7, 2017 (ICSSP'17),* 5 pages.
DOI: 10.1145/3084100.3084102

ICSSP'17, Paris, France

DOI: 10.1145/3084100.3084102

1 INTRODUCTION

Evolution and refinement of requirements is guiding the software system development process. Requirement specifications, mostly documented in natural language, are refined with additional details of design and implementation information as a software project move forwards in its development life cycle. An important task in software development process is *requirements traceability*, which is concerned with linking requirements in which one is a *refinement* of the other. Being able to establish traceability links allows stakeholders to find the origin of each requirement and track every change that has been made to it, and ensures the continuous understanding of the problem that needs to be solved so that the right system is delivered.

In practice, one is given a set of high-level (coarse-grained) requirements and a set of low-level (fine-grained) requirements, and requirements traceability aims to find for each high-level requirement all the low-level requirements that refine it. Note that the resulting mapping between high- and low-level requirements is *many-to-many*, because a low-level requirement can potentially refine more than one high-level requirement.

As an example, consider the three high-level requirements and two low-level requirements shown in Figure 1 about the well-known Pine email system. In this example, three traceability links should be established: (1) HR01 is refined by UC01 (because UC01 specifies the shortcut key for saving an entry in the address book); (2) HR02 is refined by UC01 (because UC01 specifies how to store contacts in the address book); and (3) HR03 is refined by UC02 (because both of them are concerned with the help system).

From a text mining perspective, requirements traceability is very challenging task. First, there could be abundant information irrelevant to the establishment of a link in one or both of the requirements. For instance, all the information under the Description section in UC01 is irrelevant to the establishment of the link between UC01 and HR02. Worse still, as the goal is to induce a many-to-many mapping, information irrelevant to the establishment of one link could be relevant to the establishment of another link involving the same requirement. For instance, while the Description section is irrelevant to linking UC01 and UR02, it is crucial to linking UC01 and HR01. Above all, a link can exist between a pair of requirements (HR01 and UC01) even if they do not possess any overlapping or semantically similar content words.

Virtually all existing approaches to the requirements traceability task were developed in the software engineering (SE) research

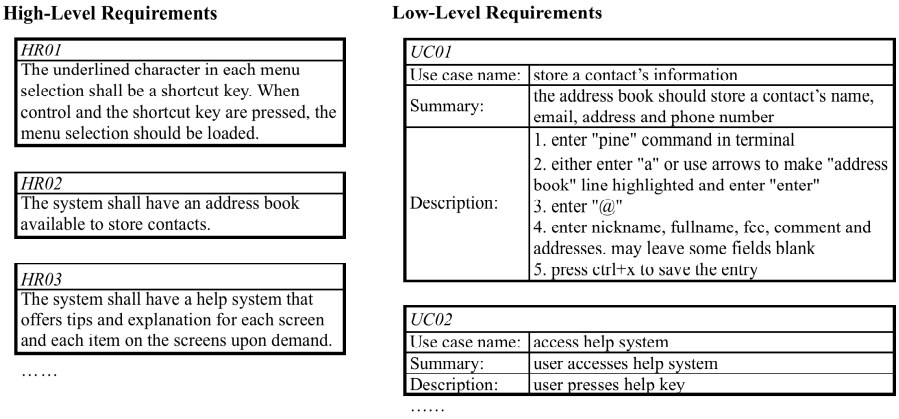

Figure 1: Samples of high- and low-level requirments.

community. Related work on this task can be broadly divided into two categories. In *manual approaches*, requirements traceability links are recovered manually by developers. *Automated approaches*, on the other hand, have relied on information retrieval (IR) techniques, which recover links based on computing the similarity between a given pair of requirements. Hence, such similarity-based approaches are unable to recover links between those pairs that do not contain overlapping or semantically similar words or phrases.

In light of this weakness, we recast requirements traceability as a supervised binary classification task, where we classify each pair of high- and low-level requirements as positive (having a link) or negative (not having a link). We represent each pair of requirements using two types of features. First, we employ *word pairs*, each of which is composed of a word taken from each of the two requirements involved. These features will enable the learning algorithm to identify both semantically similar and dissimilar word pairs that are strongly indicative of a refinement relation between the two requirements, thus overcoming the aforementioned weakness associated with similarity-based approaches.

Next, we employ features derived from an ontology hand-built by a domain expert.[1] The ontology contains only a *verb clustering* and a *noun clustering*: the verbs are clustered by the function they perform, whereas a noun cluster corresponds to a (domain-specific) semantic type.

There are at least two reasons why the ontology might be useful for identifying traceability links. First, since only those verbs and nouns that (1) appear in the training data and (2) are deemed relevant by the domain expert for link identification are included in the ontology, it provides guidance to the learner as to which words/phrases in the requirements it should focus on in the learning process.[2] Second, the verb and noun clusters provide a robust generalization of the words/phrases in the requirements. For instance, a word pair that is relevant for link identification may still be ignored by the learner due to its infrequency of occurrence. The features that are computed based on these clusters, on the other hand, will be more robust to the infrequency problem and therefore potentially provide better generalizations.

Our main contribution in this paper lies in the proposal of a *2-learner, ontology-based* approach to the task of traceability link prediction, where, for the sake of robustness, we train *two* classifiers to separately exploit the word-pair features and the ontology-based features. Results on a traceability dataset involving the Pine domain reveal that our use of two learners and the ontology-based features are both key to the success of our approach: it significantly outperforms not only a supervised baseline system that uses only word pairs features, but also a system that trains a single classifier over both the word pairs and the ontology-based features. Perhaps most interestingly, results do not deteriorate when the hand-built ontology is replaced with an automatically constructed ontology.

2 RELATED WORK

Automated or semi-automated requirements traceability has been exploited by many researchers . Pierce [11] designed a tool that maintains a requirements database to aid automated requirements tracing. Jackson [8] proposed a keyphrase-based approach for tracing a large number of requirements of a large Surface Ship Command System. More advanced approaches relying on information retrieval (IR) techniques, such as the *tf-idf*-based vector space model [13], Latent Semantic Indexing [5, 6, 10], probabilistic networks [3], and Latent Dirichlet Allocation [12], have been investigated, where traceability links were generated by calculating the textual similarity between requirements using similarity measures such as Dice, Jaccard, and Cosine coefficients [4]. All these methods were developed based on either matching keywords or identifying similar words across a pair of requirements. In recent years, Li [9] has studied the feasibility of employing supervised learning to accomplish this task.

3 DATASET

We employ the well known Pine system for evaluation. This dataset consists of a set of 49 (high-level) requirements and a set of 51 (low-level) use case specifications about Pine, an email system developed at the University of Washington. Out of the 2499 pairs of requirement and use case specification, only 10% (250) are considered traceability links.

[1]The sample ontology built for the Pine dataset is available at http://lyle.smu.edu/~zehengl/traceability/.

[2]These relevant words/phrases are reminiscent of *annotator rationales* [14]. However, it is important to note that we are not using them to generate additional training instances, unlike in the classical annotator rationales framework.

4 APPROACH

In this section, we describe our supervised approach.

4.1 Classifier Training

Each instance corresponds to a high-level requirement and a low-level requirement. Hence, we create instances by pairing each high-level requirement with each low-level requirement. The class value of an instance is positive if the two requirements involved should be linked; otherwise, it is negative. Since we conduct 5-fold cross-validation experiments, we randomly partition the instances into five folds of roughly the same size, training only four folds and evaluate on the remaining fold in each fold experiment. Each instance is represented using seven types of features, as follows.

1. Same words. We create one binary feature for each word w appearing in the training data. Its value is 1 if w appears in *both* requirements in the pair under consideration. Hence, this feature type contains the subset of the word pair features mentioned earlier where the two words in the pair are the same.

2. Different words. We create one binary feature for each word pair (w_i, w_j) collected from the training instances, where w_i and w_j are non-identical words appearing in a high-level requirement and a low-level requirement respectively. Its value is 1 if w_i and w_j appear in the high-level and low-level pair under consideration, respectively. Hence, this feature type contains the subset of the word pair features where the two words in the pair are different.

3. Verb pairs. We create one binary feature for each verb pair (v_i, v_j) collected from the training instances, where (1) v_i and v_j appear in a high-level requirement and a low-level requirement respectively, and (2) both verbs appear in the ontology. Its value is 1 if v_i and v_j appear in the high-level and low-level pair under consideration, respectively. Using these verb pairs as features may allow the learner to focus on verbs that are relevant to traceability prediction.

4. Verb group pairs. For each verb pair feature described above, we create one binary feature by replacing each verb in the pair with its cluster id in the ontology. Its value is 1 if the two verb groups in the pair appear in the high-level and low-level pair under consideration, respectively. These features may enable the resulting classifier to provide robust generalizations in cases where the learner chooses to ignore certain useful verb pairs owing to their infrequency of occurrence.

5. Noun pairs. We create one binary feature for each noun pair (n_i, n_j) collected from the training instances, where (1) n_i and n_j appear in a high-level requirement and a low-level requirement respectively, and (2) both nouns appear in the ontology. Its value is computed in the same manner as the verb pairs. These noun pairs may help the learner to focus on verbs that are relevant to traceability prediction.

6. Noun group pairs. For each noun pair feature described above, we create one binary feature by replacing each noun in the pair with its cluster id in the ontology. Its value is computed in the same manner as the verb group pairs. These features may enable the classifier to provide robust generalizations in cases where the learner chooses to ignore certain useful noun pairs owing to their infrequency of occurrence.

7. Dependency pairs. In some cases, the noun/verb pairs may not provide sufficient information for traceability prediction. For example, the verb pair feature (*delete*, *delete*) is suggestive of a positive instance, but the instance may turn out to be negative if one requirement concerns deleting messages and the other concerns deleting folders. As another example, the noun pair feature (*folder*, *folder*) is suggestive of a positive instance, but the instance may turn out to be negative if one requirement concerns creating folders and the other concerns deleting folders.

In other words, we need features that encode the verbs and nouns in isolation but the relationship between them. To do so, we first parse each requirement using the Stanford dependency parser [7], and collect each noun-verb pair (n_i,v_j) connected by a dependency relation. We then create binary features by pairing each related noun-verb pair found in a high-level training requirement with each related noun-verb pair found in a low-level training requirement. The feature value is 1 if the two noun-verb pairs appear in the pair of requirements under consideration. To enable the learner to focus on learning from relevant verbs and nouns, only verbs and nouns that appear in the ontology are used to create these features.

We employ LIBSVM [2] as the learning algorithm for training a binary SVM classifier on the training set. We use a linear kernel, tuning the C value (the regularization parameter) to maximize F-score on the development (dev) set. All other learning parameters are set to their default values.

To improve performance, we perform feature selection (FS) using the backward elimination algorithm [1]. Starting with all seven feature types, the algorithm iteratively removes one feature type at a time until only one feature type is left. Specifically, in each iteration, it removes the feature type whose removal yields the largest F-score on the dev set. We picked the feature subset that achieving the largest F-score on the dev set over all iterations.

Note that tuning the C value (from libSVM) and selecting the feature subset both require the use of a dev set. In each fold experiment, we reserve one fold for development and use the remaining three folds for training classifiers. We jointly tune the C value and select the feature subset to maximize F-score on the dev set.

4.2 Two Extensions

Next, we present two extensions to our supervised approach.

4.2.1 Employing Two Views. Our first extension involves splitting our feature sets into two *views* (i.e., disjoint subsets) and training one classifier on each view. To motivate this extension, recall that the ontology is composed of words and phrases that are deemed relevant to traceability prediction according to a SE expert. In other words, the (word- and cluster-based) features derived from the ontology (i.e., features 3–7 in our feature set) are sufficient for traceability prediction, and the remaining features (features 1 and 2) are not needed according to the expert. While some of the word pairs that appear in features 1 and 2 also appear in features 3–7, most of them do not. If these expert-determined irrelevant features are indeed irrelevant, then retaining them could be harmful for classification because they significantly outnumber their relevant counterparts. However, if some of these features are relevant (because some relevant words are missed by the expert, for instance), then removing them would not be a good idea either.

Our solution to this dilemma is to divide the feature set into two views. Given the above discussion, a natural feature split would involve putting the ontology-based features (features 3–7) into one view and the remaining ones (features 1–2) into the other view.

Then we train one SVM classifier on each view as before. During test time, we apply both classifiers to a test instance, classifying it using the prediction associated with the higher confidence value.[3] This setup would prevent the expert-determined irrelevant features from affecting the relevant ones, and at the same time avoid totally discarding them in case they do contain some relevant information.

A natural question is: why not simply use backward elimination to identify the irrelevant features? While we believe FS can help, it may not be as powerful as one would think because (1) backward elimination is greedy; and (2) the features are selected using a fairly small set of instances (i.e., the dev set) and may therefore be biased towards the dev set.

In fact, we view our 2-learner setup and FS as *complementary* rather than *competing* solutions to our dilemma. In particular, we will employ FS in the 2-learner setup: when training the classifiers on the two views, we employ backward elimination in the same way as before, removing the feature type (from one of the two classifiers) whose removal yields the highest F-score on the dev set in each iteration.

4.2.2 Learning the Ontology. An interesting question is: can we learn instead of hand-build the ontology? Not only is this question interesting from a research perspective, it is of practical relevance: even if a domain expert is available, hand-constructing the ontology is a time-consuming and error-prone process. Below we describe the steps we propose for ontology learning, which involves producing a verb clustering and a noun clustering.

Step 1: Verb/Noun selection. We select the nouns, noun phrases (NPs) and verbs in the training set to be clustered. Specifically, we select a verb/noun/NP if (1) it appears more once in the training data; (2) it contains at least three characters (thus avoiding verbs such as *be*); and (3) it appears in the high-level but not the low-level requirements and vice versa.

Step 2: Verb/Noun representation. We represent each noun/NP/verb as a feature vector. Each verb v is represented using the set of nouns/NPs collected in Step 1. The value of each feature is binary: 1 if the corresponding noun/NP occurs as the direct or indirect object of v in the training data (as determined by the Stanford dependency parser), and 0 otherwise. Similarly, each noun n is represented using the set of verbs collected in Step 1. The value of each feature is binary: 1 if n serves as the direct or indirect object of the corresponding verb in the training data, and 0 otherwise.

Step 3: Clustering. To produce a verb clustering and a noun clustering, we cluster the verbs and the nouns/NPs separately. We experiment with two clustering algorithms. The first one, which we refer to as *Simple*, is the classical single-link algorithm. Single-link is an agglomerative algorithm where each object to be clustered is initially in its own cluster. In each iteration, it merges the two most similar clusters and stops when the desired number of clusters is reached. The second clustering algorithm is motivated by the following observation. We could produce a better verb clustering if each verb were represented using noun *categories* rather than nouns/NPs, because there is no need to distinguish between the nouns in the same category in order to produce the verb clusters we desire. Similarly, we could produce a better noun clustering if each noun were represented using verb categories.

[3]To compute the confidence value associated with a prediction, we take the absolute distance of the underlying test instance from the hyperplane.

In practice, we do *not* have the noun and verb categories (because they are what the clustering algorithm is trying to produce). However, we can use the (partial) verb clusters produced during the verb clustering process to improve noun clustering and vice versa. This motivates our *Interactive* clustering algorithm. Like *Simple*, *Interactive* is also a single-link clustering algorithm. Unlike *Simple*, which produces the two clusterings separately, *Interactive* interleaves the verb and noun clustering processes, as described below.

Initially, each verb and each noun is in its own cluster. In each iteration, we (1) merge the two most similar verb clusters; (2) update the noun's feature representation by merging the two verb features that correspond to the newly formed verb cluster[4]; (3) merge the two most similar noun clusters using this updated feature representation for nouns; (4) update the verb's feature representation by merging the two noun features that correspond to the newly formed noun cluster. As in *Simple*, *Interactive* terminates when the desired number of clusters is reached.

For both clustering algorithms, we compute the similarity between two objects by taking the dot product of their feature vectors. Since we are using single-link clustering, the similarity between two clusters is the similarity between the two most similar objects in the two clusters.

Since we do not know the number of clusters to be produced *a priori*, we produced three noun clusterings and three verb clusterings (with 10, 15, and 20 clusters each). We then select the combination of noun clustering, verb clustering, the C value, and the feature subset that maximizes F-score on the dev set, and apply the resulting combination on the test set.

5 EVALUATION

5.1 Experimental Setup

We employ as our evaluation measure F-score, which is the unweighted harmonic mean of recall and precision. Recall is the percentage of links in the gold standard that are recovered by our system. Precision is the percentage of links recovered by our system that are correct. We preprocess each document by removing stopwords and stemming the remaining words. All results are obtained via 5-fold cross validation.

5.2 Results and Discussion

5.2.1 Baseline Systems. We present two unsupervised and two supervised baselines.

Baseline 1: Tf.idf. Motivated by previous work, we employ *tf.idf* as our first unsupervised baseline. Each document is represented as a vector of unigrams. The value of each feature is its *tf.idf* value. Cosine is used to compute the similarity between two documents. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive. We tested thresholds from 0.1 to 0.9 with an increment of 0.1 and report results using the best threshold, essentially giving an advantage to it in the performance comparison. As we can see in row 1 of Table 1, it achieves an F-score of 54.5%.

Baseline 2: LDA.. Also motivated by previous work, we employ LDA as our second unsupervised baseline. We train an LDA on our data to produce n topics (where n=10, 20, . . ., 60). We then use

[4]This will reduce the number of features by one. The value of the "merged" feature will be the disjunction of the values of the original features.

Table 1: Five-fold cross-validation results.

	System	Feature Selection?	Recall	Prec.	F-score
	Baseline Systems				
1	Tf.idf	N/A	73.6	43.3	54.5
2	LDA	N/A	30.4	39.2	34.2
3	Features 1&2	No	50.0	66.5	57.1
		Yes	62.4	73.9	67.7
4	Features 1&2 + LDA	No	50.4	67.0	57.5
		Yes	66.0	72.4	69.0
	Our Approach				
5	Single learner +manual clusters	No	54.0	73.0	62.1
		Yes	66.8	79.1	72.5
6	Single learner +induced clusters	No	53.2	73.5	61.7
		Yes	65.6	78.1	71.3
7	Two learners +manual clusters	No	61.6	84.6	71.3
		Yes	68.4	84.2	75.5
8	Two learners +induced clusters	No	62.8	81.8	71.0
		Yes	71.2	84.0	77.1

the n topics as features for representing each document, where the value of a feature is the probability the document belongs to the corresponding topic. Cosine is used as the similarity measure. Any pair of requirements whose similarity exceeds a given threshold is labeled as positive. We tested thresholds from 0.1 to 0.9 with an increment of 0.1 and report results using the best threshold, essentially giving an advantage to it in the performance comparison. As we can see in row 2 of Table 1, it achieves an F-score of 34.2%.

Baseline 3: Features 1 and 2. As our first supervised baseline, we train a SVM classifier using only features 1 and 2 (all the word pairs). As we can see from row 3 of Table 1, it achieves F-scores of 57.1% (without FS) and 67.7% (with FS). These results suggest that FS is indeed useful.

Baseline 4: Features 1, 2, and LDA.. As our second supervised baseline, we augment the feature set used in Baseline 3 with the LDA features used in Baseline 2 and then train a SVM classifier. We select the best n (number of topics) using the dev set. As we can see in row 4 of Table 1, this is the best of the four baselines: it significantly outperforms Baseline 3 regardless of whether feature selection is performed[5], suggesting the usefulness of the LDA features.

5.2.2 Our Approach. Next, we evaluate our 2-learner, ontology-based approach. In the single-learner experiments, a classifier is trained on the seven features described in Section 4.1, whereas in the 2-learner experiments, these seven features are split as described in Section 4.2.

Setting 1: Single learner, manual clusters. As we can see in row 5 of Table 1, this classifier significantly outperforms the best baseline (Baseline 4): F-scores increase by 4.6% (without FS) and 3.5% (with FS). Since the only difference between this and Baseline 4 lies in whether the LDA features or the ontology-based features are used, these results seem to suggest that features formed from the clusters in our hand-built ontology are more useful than the LDA features.

Setting 2: Single learner, induced clusters. As we can see from row 6 of Table 1, this classifier performs statistically indistinguishably from the one in Setting 1. This is an encouraging result: it shows that even when features are created from induced rather than manual clusters, performance does not significantly drop regardless of whether FS is performed.

[5]All significance tests are two-tailed paired t-tests ($p < 0.05$).

Setting 3: Two learners, manual clusters. As we can see from row 7 of Table 1, this classifier performs significantly better than the one in Setting 1: F-scores increase by 9.2% (without FS) and 3.0% (with FS). As the two settings differ only w.r.t. whether one or two learners are used, the improvements suggest the effectiveness of our 2-learner framework.

Setting 4: Two learners, induced clusters. As we can see from row 8 of Table 1, this classifier performs significantly better than the one in Setting 2: F-scores increase by 9.3% (without FS) and 5.8% (with FS). It also performs indistinguishably from the one in Setting 3. Taken together, these results suggest that (1) our 2-learner framework is effective in improving performance, and (2) features derived from induced clusters are as effective as those from manual clusters.

Overall, these results show that (1) our 2-learner, ontology-based approach is effective, and (2) feature selection consistently improves performance.

6 CONCLUSIONS

We investigated a 2-learner, ontology-based approach to supervised traceability prediction. Results showed that (1) our approach is effective: in comparison to the best baseline, relative error reduces by 25.9%; and (2) results obtained via induced clusters were as competitive as those obtained via manual clusters.

REFERENCES

[1] Avrim Blum and Pat Langley. 1997. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence* 97, 1–2 (1997), 245–271.

[2] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 27.

[3] Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. 2005. Utilizing supporting evidence to improve dynamic requirements traceability. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on.* IEEE, 135–144.

[4] Johan Natt Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, and Joachim Karlsson. 2002. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering* 7, 1 (2002), 20–33.

[5] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 16, 4 (2007), 13.

[6] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. 2009. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering* 14, 1 (2009), 57–92.

[7] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation.* 449–454.

[8] Justin Jackson. 1991. A keyphrase based traceability scheme. In *Tools and Techniques for Maintaining Traceability During Design, IEE Colloquium on.* IET, 2–1.

[9] Zeheng Li, Mingrui Chen, LiGuo Huang, and Vincent Ng. 2015. Recovering Traceability Links in Requirements Documents. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning.* 237–246.

[10] Marco Lormans and Arie Van Deursen. 2006. Can LSI help reconstructing requirements traceability in design and test?. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on.* IEEE, 10–pp.

[11] Robert A Pierce. 1978. A requirements tracing tool. *ACM SIGSOFT Software Engineering Notes* 3, 5 (1978), 53–60.

[12] Daniel Port, Allen Nikora, Jane Huffman Hayes, and LiGuo Huang. 2011. Text mining support for software requirements: Traceability assurance. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on.* IEEE, 1–11.

[13] Senthil Karthikeyan Sundaram, Jane Huffman Hayes, and Alexander Dekhtyar. 2005. Baselines in requirements tracing. *ACM SIGSOFT Software Engineering Notes* 30, 4 (2005), 1–6.

[14] Omar Zaidan, Jason Eisner, and Christine Piatko. 2007. Using "Annotator Rationales" to Improve Machine Learning for Text Categorization. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference.* 260–267.