Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations

Tobias Hey, Fei Chen, Sebastian Weigelt, Walter F. Tichy
Karlsruhe Institute of Technology (KIT)
Institute for Program Structures and Data Organization
Karlsruhe, Germany
hey@kit.edu, Fe.Ch@web.de, weigelt@kit.edu, tichy@kit.edu

***Abstract*—Traceability information is a fundamental prerequisite for many essential software maintenance and evolution tasks, such as change impact and software reusability analyses. However, manually generating traceability information is costly and error-prone. Therefore, researchers have developed automated approaches that utilize textual similarities between artifacts to establish trace links. These approaches tend to achieve low precision at reasonable recall levels, as they are not able to bridge the semantic gap between high-level natural language requirements and code.**

We propose to overcome this limitation by leveraging fine-grained, method and sentence level, similarities between the artifacts for traceability link recovery. Our approach uses word embeddings and a Word Mover's Distance-based similarity to bridge the semantic gap. The fine-grained similarities are aggregated according to the artifacts structure and participate in a majority vote to retrieve coarse-grained, requirement-to-class, trace links.

In a comprehensive empirical evaluation, we show that our approach is able to outperform state-of-the-art unsupervised traceability link recovery approaches. Additionally, we illustrate the benefits of fine-grained structural analyses to word embedding-based trace link generation.

***Index Terms*—Traceability, Traceability Link Recovery, Requirements Engineering, Word Embeddings, Natural Language Processing, Word Movers Distance**

I. Introduction

Requirements traceability empowers many down stream tasks of the software development and maintenance process. The availability of traceability links between any type of requirement and source code boosts the performance of tasks such as change impact analysis, requirements validation, or software reusability analysis [1]. For example, connecting requirements to source code elements (that implement them) provides insight into what has been implemented and where; it also reveals semantic relations between source code elements. In safety-critical systems traceability is essential to demonstrate the functionality of the system as prescribed by regulatory bodies [2]. However, due to the high manual effort for creating and maintaining the trace links, traceability information is absent or incomplete in most software projects.

Most approaches that automatically recover trace links make use of information retrieval (IR) [3]–[9] or machine learning (ML) [10]–[12] techniques. Since both code and requirements consist of sequences of (natural language) words (for the greater part at least), they essentially quantify the textual similarities of the artifacts. These approaches obtain relatively low precision at reasonable recall levels. As they rely upon the syntactical features of words, they struggle to determine the similarity of artifacts that are not syntactically related. These approaches ignore the different levels of abstraction in the wording of source code and requirements. Apparently, there is a semantic gap between the different artifact types that must be bridged.

DOI 10.1109/ICSME52107.2021.00008

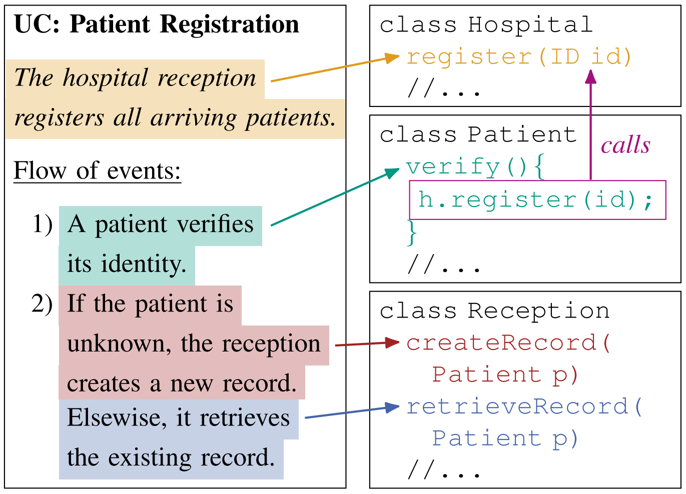

UC: Patient Registration
The hospital reception registers all arriving patients.
Flow of events:
1) A patient verifies its identity.
2) If the patient is unknown, the reception creates a new record.
Elsewise, it retrieves the existing record.
class Hospital
register(ID id)
//...
class Patient
verify(){
h.register(id);
}
//...
calls
class Reception
createRecord(
Patient p)
retrieveRecord(
Patient p)
//...

Fig. 1. Example of fine-grained relations between a use case description and source code elements.

In order to take semantics into account other approaches use topic models [13] or word embeddings [10], [11], [14] but miss the opportunity to utilize fine-grained relations between artifacts. Instead, they create coarse-grained representations of artifacts, mostly by combining all word embeddings of the resp. artifact (e.g., calculating an average vector). However, such combinations may be misleading if semantically unrelated words or too many aspects are aggregated. If we take a look at the example in Figure 1 simply aggregating the contained words in the use case description or the classes might result in vector representations in between the described aspects. This makes it more difficult to connect the representations to their corresponding counterparts. If the artifacts are interpreted in a more fine-grained manner, similarities between the elements can be derived more clearly. Additionally, coarse-grained approaches miss the opportunity to utilize fine-grained inter-artifact relations, such as the call dependency between the `verify` and `register` methods, which for both artifacts hints at their shared relevance for the use case description.

To address this issue, we propose to use word embeddings in a more fine-grained manner to recover requirements-to-code trace links. We use fastText word embeddings [15] to calculate similarities between methods and requirement segments (mostly sentences) and aggregate the results by applying majority votes to retrieve class-level trace links. Our approach makes use of structural information in source code (e.g., call graphs) and requirements (i.e., common use case templates). Compared to related work, our approach obtains higher precision values (for similar recall values). Thus, the use of fine-grained relations leads to automated traceability link recoveries of higher quality (regarding F_1-score). Our contributions are:

1) A novel approach for traceability link recovery using fine-grained requirements-to-code relations and word embeddings.
2) An empirical study on the effect of different structural information in word embedding-based traceability link recovery.
3) A comparison to state-of-the-art traceability link recovery approaches.

We publish the source code in our supplementary material [16].

II. Related Work

Research on automated requirements traceability has been conducted since the 1990s and achieved its first breakthrough with IR techniques. Techniques such as vector space models (VSM) [3], latent semantic indexing (LSI) [4], [17], or latent dirichlet allocation (LDA) [13] have been used.

One way to improve results is to combine several techniques: Gethers et al. [5] use a combination of VSM, Jensen-Shannon models and relational topic models. Lohar et al. [6] combine different preprocessing, dictionary building, and similarity calculation techniques by applying a genetic algorithm on an initial set of trace links. Another way to improve results is to integrate additional knowledge sources: Hayes et al. [7], [18] make use of domain specific information from thesauri and Zou et al. [8] weight terms higher that occur in the requirement glossaries. Others integrate domain and world knowledge via ontologies [19], [20]. Recently, Moran et al. [9] employ a hierarchical Bayesian network to combine multiple IR- and ML-based textual similarities with developer feedback and transitive trace links. Their tool COMET is able to select reasonable combinations without the need of initial trace links.

IR-based approaches constituted a major step towards automated traceability but still obtain relatively low precision at reasonable recall levels. As most of them are based on the probability of words and tokens, the approaches struggle to determine similarities between syntactically unrelated artifacts.

Another research direction considers information about the structure and dependencies of source code to determine classes and chunks of related code. Panichella et al. [21] have shown that utilizing call and inheritance dependencies is beneficial for traceability link recovery. Kuang et al. [22] additionally employ method data dependencies and show that they provide complementary information to call dependencies. Florez [23] proposed using a more fine-grained view of the artifacts by identifying functional constraints in requirements and source code, but has not reported any results yet.

Considering the underlying semantics of the requirements presents another approach to close the semantic gap between the artifacts. Approaches integrate different semantic information into IR methods [24] or incorporate semantics implicitly by utilizing word embeddings. Guo et al. [10] use word embeddings for requirements-to-requirements traceability in a supervised deep learning setting. Mills et al. [25] propose another supervised approach called TRAIL to traceability link recovery that incorporates several IR-, text quality- and artifact-based features in a random forest classifier. One drawback of supervised approaches is that they need a training set of gold standard trace links. To cope with that, Mills et al. [12] investigate with ALCATRAL how active learning can decrease the amount of needed training data. Zhao et al. [11] apply word embeddings in a supervised setting. Their approach WQI uses the cosine similarity and a learning-to-rank technique. Chen et al. [14] use document embeddings in combination with sequential semantics to incorporate sequential information as well. Their approach S2Trace is unsupervised. Thus, it does not require a training set of trace links.

Embedding-based approaches still achieve low F_1-scores. They are unable to grasp fine-grained relations between the artifacts as they aggregate information on the artifact level. Additionally, they utilize word embedding models trained on the dataset that might be too small to learn all relevant relations between the words. Therefore, we investigate how fine-grained word embedding-based similarities can improve results for unsupervised requirements-to-code traceability link recovery.

III. Background

Most IR-based techniques represent natural language as high-dimensional vectors using techniques such as tf-idf or Bag-of-Words. Since the vocabulary size is usually large in real-world applications, these vectors are very sparse. Furthermore, they do not take relations between words into account.

To overcome these limitations word embeddings were introduced as a vector representation with a fixed number of dimensions that is distinctly lower than the vocabulary size (typically between 100 and 300). The idea of word embeddings is to represent words by the context they occur in. Thus, the embeddings of words that occur in the same contexts are close in the vector space. To get vector representations that have a lower number of dimensions an embedding matrix is learned in an unsupervised intermediate task on a text corpus. The most common intermediate tasks are continuous Bag-of-Words (CBOW) and continuous skip-gram with negative sampling as introduced by *word2vec* [26]. For both the text is scanned with a sliding window. The CBOW task involves the prediction of the word in the center of the sliding window (by taking the other words in the window into account). In the skip-gram task, a classifier is trained to predict whether a certain word of positive and negative samples is one of the words in its context window (for each word in the window center). Each row of the embedding matrix represents the word embedding of one of the vocabulary words.

One drawback of this approach is that it is unable to represent a word it has not seen during training. The *fastText* [27] approach overcomes this limitation: It also creates embeddings for the character n-grams in the words. Therefore, it is able to provide embeddings for unseen words or subwords based on the contained character n-grams. The rationale behind this approach is that in many cases removing pre- or suffixes of words does not affect their meaning and words that share character n-grams might be related. For example, *fastText* represents the word *prefix* by the sum of the vector representations of its character trigrams `<pr`, `pre`, `ref`, `efi`, `fix`, `ix>` and the special sequence `<prefix>`. The symbols < and > are special symbols to represent the boundaries of words. *fastText* uses all character n-grams of size $n = 3$ to 6.

A word embedding model that can represent unseen words allows the usage of models for texts that contain words that were not part of the training corpus. In requirements engineering and source code understanding this is beneficial since the artifacts often include technical terms and abbreviations.

A. Similarity Measures

Word embeddings are designed to represent words that occur in the same contexts close to each other in the vector space. Thus, they can be used to calculate similarities between words and documents. The two most commonly used similarity measures for vector spaces are the euclidean distance and cosine similarity. The euclidean distance is calculated between two vectors u and v as follows:

$$euclid(u, v) = \|u - v\|_2 \tag{1}$$

The euclidean distance can be interpreted as similarity measure: the smaller the distance the more similar the vectors.

The cosine similarity is independent of the order of magnitude of the vectors; it determines the proximity by calculating the angle ϕ between the vectors. This replicates the inner product of the two vectors normalized to length one.

$$cosSim(u, v) = \frac{u \cdot v}{\|u\|_2 \, \|v\|_2} = \cos\phi \tag{2}$$

The similarity of documents can be measured with the cosine or the euclidean similarity as well. Therefore, the words (represented as vectors) in the documents are aggregated. One approach is to average all word vectors in each document for comparison; another is to calculate the similarities between all pairs of vectors (of two documents) and interpreting the maximum/minimum as the document similarity. However, these approaches may oversimplify the relations between the words by merging different aspects or merely provide upper and lower bounds but do not consider the shared meaning.

The Word Mover's Distance (WMD) [28] overcomes these limitations by measuring the semantic distance of two documents. It interprets the euclidean distance of each pair of word embeddings in the documents as their respective semantic similarity. Then the semantic distance between the two documents is calculated as the minimum cumulative euclidean distance required to map each word (embedding) of document A to a word (embedding) of document B. So each word of document A is mapped to the nearest (most similar) word of document B and the documents similarity is the cumulative distance of these mappings.

IV. Inferring Trace Links with Fine-grained Word Embedding Similarities

Our approach **F**ine-grained **T**raceability **L**ink **R**ecovery (FTLR) automatically provides requirements-to-code trace links in an unsupervised setting. This means it recovers trace links without the need for information about existing links (as would be the case if applied to a project that does not have traceability information). Without the precondition of having a training set of existing trace links, downstream tasks on any project can benefit (that otherwise would not have access to this information). We propose to use fine-grained word embedding-based similarities. Artifacts often comprise multiple aspects, such as the flow of events in a use case description or classes with methods serving different purposes. We assume that a fine-grained approach enables us to distinguish these aspects more properly. Further, the use of word embeddings enables us to calculate similarities between artifacts that are not only based on syntactic similarities but on semantic relatedness.

Our approach aggregates the fine-grained similarities using majority votes. Unrelated candidates are filtered and an overview of the artifacts aspects is provided. Additionally, we exploit the structural information of source code, such as method call dependencies and relations between methods and classes, to improve candidate ranking. Similarly, we use common use case templates to utilize structures in requirements.

In the following sections we first present the underlying approach of our fine-grained method. Then, we investigate additional information that may improve our basic approach.

A. Fine-grained Traceability Link Recovery

Our approach operates on requirement sentences and (source code) methods as smallest units of representation (hereinafter called artifact elements). The rationale behind this decision is that (well-written) requirement sentences express a single cohesive semantics and the same should hold true for methods. We do not consider smaller units such as the contents of the method body, as they mainly contain terms that contribute information on the implementation level. In a preliminary experiment they introduced more noise than valuable information.

FTLR establishes trace links between requirements and source code using relations between sentences and methods; it applies the following three steps:

1) Represent the artifact elements using word embeddings.
2) Calculate similarities between artifact elements.
3) Aggregate the fine-grained relations to provide trace links.

1) Representation: FTLR represents each element of both artifact types (sentences and methods) as so-called Bag-of-Embeddings (BoE). BoE are sets of word embeddings that represent all words of the respective artifact element. For requirements, we use all words of a sentence for the BoE. On the code side, for each method, we use the method name, parameter names and types, and the name of the containing class. FTLR retrieves the vector representation of each word from a pre-trained *fastText* word embedding model (trained on *Wikipedia* and the *CommonCrawl* dataset) [29]. The *CommonCrawl* dataset consists of all kinds of texts crawled from the web. We use a general purpose word embedding model because it covers most words, relations and domains. However, this type of model may undervalue relations from specific contexts by aggregating the relations of all senses of a word. We leave the analyses of the potential of domain-specific models to future work. During preprocessing we remove stop words with *NLTK* [30], lemmatize words with *spaCy* [31] and remove non-textual elements, such as hyperlinks, non character tokens and numbers. For source code elements we additionally split tokens at camel case, remove programming language specific stop words, and filter words according to their length (removing tokens with one or two characters)[1]. In addition to programming language specific stop words, we filter common terms in programming, such as *get*, *set*, *array*, and *exception*.

Not every method contributes equally to the purpose of its class. The purpose of a class should be described by the interface of the class, its publicly visible elements. Besides, private methods are often merely helper methods and are too implementation specific to be valuable for traceability link recovery. Therefore, we only consider methods that are publicly visible. This approach may lead to empty representations for some classes. In these cases FTLR uses the class name as an artifact element instead of the methods. Note that in all other cases the tokens of the class name are added to the BoE of a method. That way FTLR is able to distinguish methods with the same name from different classes. For example, two classes `ShoppingCart` and `Stock` have a method `add(Item)` that can be easily distinguished by adding the class name.

2) Similarity Calculation: With the fine-grained representations of the artifacts at hand we can calculate the similarity of the elements. As we use the same word embedding model for all representations, we are able to use its characteristics. Words that occur in the same contexts have embeddings that are close-by (regarding their cosine angle and euclidean distance).

In subsection III-A we discussed different similarity measures for vector representations. Related work [6], [11], [18] used the cosine similarity for this purpose. However, we would need to aggregate the word embeddings in the BoE to retrieve a single representation for the respective element or use the maximum similarity. The former is based on the assumption that averaging word embeddings of an artifact element results in a representative vector. If multiple aspects/purposes are present,

[1]The rationale behind the word length filter is that token with less than three characters most likely are abbreviations that are difficult to interpret.

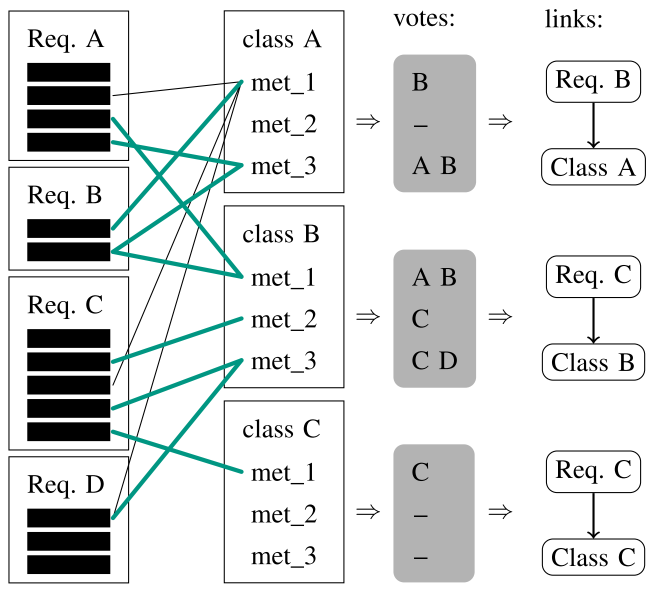


Fig. 2. Overview of FTLR approach, displaying fine-grained relations between class methods and requirements, each method's votes and the resulting trace links. Thick lines indicate similarities above the majority threshold and further similarities are omitted to ensure readability.

aggregating word embeddings results in a vector somewhere in between the aspects that might even represent a utterly different aspect. The latter is incapable to grasp semantics that result from the combination of the words. Therefore, we propose to use WMD as similarity metric to express fine-grained similarities instead. It grasps the relatedness of two BoEs by calculating their minimum cumulative distance. Thus, WMD considers all words in the BoE as equally important and can grasp the semantics of combinations of words.

For each pair of fine-grained source and target artifact elements we define their similarity to be their WMD. A smaller value means a higher similarity here. The possible values for WMD with normalized word embedding vectors (max. length = 1) reside in the range $[0, 2]$. However, we normalize the values by projecting them to $[0, 1]$.

3) Aggregation: In general, not every artifact element has a counterpart. For example, many methods are merely helper methods and do not provide any insights into the purpose of the class. Vice versa, some parts of the requirements might not be represented in the code. However, there are many fine-grained relations between requirements and code our approach may use. Thus, the task is to identify those fine-grained relations that contribute the most to the overall relatedness of the artifacts. We then aggregate these relations to establish trace links on the artifact-level (coarse-grained).

Since an artifact may convey multiple aspects/purposes, aggregating the similarities by averaging them or simply using the highest similarity is not meaningful. Instead, we use the approach depicted in Figure 2. We start with the fine-grained relations between each requirement and source code element. For each source code element (ce) FTLR retrieves the most similar element of a requirement (R) (the relations on the left side of Figure 2)[2]:

$$sim(ce, R) = \min_{re \in R} wmd(ce, re) \tag{3}$$

This relation serves as the representative for the similarity between a source element (fine-grained) and an entire requirement (coarse-grained). Thus, the relation of a method to a requirement is defined by the most similar relation between them. Next FTLR removes those method-to-requirement relations that have a low similarity, since they do not contribute to the relatedness of the class. Based on a preliminary study on a subset of the eTour, iTrust, SMOS and eAnci datasets (see subsection V-A) we choose a threshold of 0.59 for this purpose[3]. We call this threshold the majority threshold. In Figure 2 these relations are drawn as thick green lines. To aggregate the relations of all elements in a class FTLR uses a majority vote. Each method votes for the requirements it is related to after applying the majority threshold. For example, in Figure 2 method one of class A only votes for requirement B as its most similar relations to the other requirements do not undercut the threshold. Finally, FTLR creates a trace link candidate for those requirements that got the most votes per class. The rationale behind the majority vote is that we aim at using the most prevalent purposes of a class only to increase precision. In the example a trace link candidate is created for requirement B in class A, requirement C in class B and requirement C in class C. Note that if multiple requirements receive the same number of votes we create trace link candidates for each.

The trace link candidates are filtered by another threshold to retrieve the final trace links. The idea is that a prevalent purpose (a voted trace link candidate) should have at least one contributing relatedness that is higher than this final threshold (0.44). Therefore, FTLR retrieves the most similar element that voted for the trace link candidate. If the similarity is lower (a higher WMD) than the threshold, the link is removed. We also experimented with the average similarity of all voters, but the maximum performed best. Note that, FTLR applies two thresholds to be less restrictive before voting; thus, it determines the best link candidate with a higher chance while still ensuring a minimum relatedness after voting.

B. Method Comments

Another source of information on the purpose of a method is its (documentation) comment. So far FTLR only utilizes the method's signature. As method comments may vary in quality and might not be present at all, we regard this information as an optional source of information FTLR can exploit.

We incorporate the method comments into FTLR by adding the embedding representation of the words in a method's comment to the BoE of the method. Thus, the information contributes to the fine-grained similarities between the method and the requirement elements. We preprocess the comments in the way we treat the other source code elements and additionally remove special tags, such as Javadoc tags. For structured comments, such as Javadoc comments, we only use the descriptive part of the comment, ignoring parameter, return or throws descriptions, since the covered information and wording resembles the descriptive part in the majority of cases. We label variants of FTLR that use method comments as source of information with *+mc*.

[2]Note that each source code element has such relations to each requirement. However, we only depict all relations for the first source code element in Figure 2 for the sake of clarity.

[3]Thus, only relations are considered that have a WMD lower than 0.59. Keep in mind that a lower WMD represents a higher similarity.

C. Method Call Dependencies

Previous approaches have shown that interpreting method call dependencies can improve traceability link recovery [21], [22]. The methods that call the method in question and all methods that are called by it may contribute to the interpretation of the purpose; the calling and called methods can be regarded as the context of the method in question. We enable FTLR to utilize this information by integrating the relatedness of calling and called methods into the relatedness score of a method. FTLR establishes this context by aggregating the mean similarity of all calling and called methods and the similarity score of the method. We use the mean similarity to normalize the contribution. As we want the method in question to still be the most important source of information (regarding its purpose), FTLR uses a weighted aggregation.

Like the consideration of comments, the use of called/-ing methods is optional. If this option is set, FTLR updates the similarity scores (sim) for each method as follows. We weight the similarity of the current method (cm) to a requirement (R) with 0.9 and the mean similarity of all called/-ing methods (CD) with 0.1:

$$simCD(cm, R) = 0.9\ sim(cm, R) + 0.1 \frac{\sum_{m \in CD} sim(m, R)}{|CD|} \tag{4}$$

We label variants of FTLR that use method call dependencies as additional source of information with *+cd*.

D. Use Case Templates

In this work we regard use case descriptions and plain requirements as source artifacts and refer to them as requirements. Since these requirements are natural language documents, they have a structure. This structure may be defined by discourse structure elements, such as sentences or headlines, or they are shaped according to a template. The structure can indicate how certain elements of the requirements have to be interpreted. For example, a use case description can include different fields of information (flow of events, actors, pre-/post-conditions, etc.).

In some cases it might be sufficient to let all (meaningful) words of an artifact contribute to its representative, as they convey the same aspect. However, in other cases, such as use case descriptions, the different elements may contain information on the overall aspect (use case name or description), certain functionalities (sentences/enumerations of flow of events) or are irrelevant for traceability link recovery (actors or some conditions). The descriptions in the flow of events

TABLE I
THE DATASETS USED FOR EVALUATION. ALL DATASETS CONSIST OF USE CASES AS SOURCE ARTIFACTS AND SOURCE CODE AS TARGET ARTIFACTS.

		Language		Number of Artifacts			∅ Methods per Class		Covered Artifacts		Baseline	
Project	Domain	Natural	Programming	Source	Target	Links	public	non-public	Source	Target	Precision	F_1
eTour	Tourism	EN/IT	Java	58	116	308	7.0	3.6	0.983	0.767	0.046	0.088
iTrust	Healthcare	EN	Java	131	226	286	6.5	0.3	0.802	0.385	0.010	0.020
SMOS	Education	IT/EN	Java	67	100	1044	3.5	2.5	1.000	0.684	0.159	0.274
eAnci	Governance	IT	Java	139	55	567	5.1	0.2	0.281	1.000	0.074	0.137

might even explicitly correspond to different parts of the source code and therefore should not be aggregated easily. Therefore, FTLR interprets the structure of common use case templates to improve traceability link recovery. FTLR retrieves the fields of the template and represents them individually following the same approach we used for full requirements. This allows us to retrieve similarities for each of the elements and use only the fields that most likely contribute information on potential links. We use the name, the description, and the flow of events (if present). As they are often not represented in the source code, we discard actors and pre-/post-conditions.

We label variants of FTLR that consider the structure of use case templates with *+uct*.

V. RESEARCH DESIGN

In order to provide insights into the performance of FTLR we perform several experiments. First, we determine the best performing variants of FTLR. Then we use these variants for comparison to state-of-the-art approaches. Thus, the experiments aim at answering the following questions:

RQ1: *To what extent does the use of method comments, call dependencies and knowledge of the structure of use cases improve the performance of a word embedding-based requirements-to-code traceability link recovery?*
Previous work has shown that structural information improves IR-based traceability link recovery approaches. We investigate the effect of different feature configurations on the performance of FTLR.

RQ2: *How do the chosen thresholds affect the performance of our approach?*
One issue of our approach is determining the best performing thresholds. Therefore, we analyse whether the defined thresholds are generally valid and measure their impact in comparison to the optimal thresholds for different projects.

RQ3: *How does a requirements-to-code traceability link recovery approach using fine-grained word embedding-based relations perform in comparison to state-of-the-art approaches?*
We investigate how an approach that uses word embeddings for fine-grained artifact similarities performs on class-level requirements-to-code traceability link recovery. Therefore, we compare our approach to state-of-the-art automated traceability link recovery approaches.

A. Datasets

To answer the research questions we make use of the four datasets shown in Table I. They are provided by the Center of Excellence for Software & Systems Traceability (CoEST) [32] and are commonly used in automated traceability link recovery [5], [6], [9]. We choose these datasets as they provide trace links between natural language requirements and object-oriented source code. Furthermore, we only use datasets with compilable source code; this facilitates the application of static code analyses. Note that iTrust is composed of Java and JSP target artifacts. However, we only consider the Java target artifacts and links here.

eTour mainly comprises natural language text written in English except for the identifiers in the source code and the names of the use cases, which are Italian. In contrast, SMOS consists of Italian text for both, use cases and source code comments. However, the identifiers are in English. To provide a consistent language for each project one of the authors translated the respective identifiers into the prevalent language by dictionary. For example, the identifier *cognome* was translated to *surname* in the eTour dataset. As our approach is based on pre-trained word embeddings that depict similarities between words, it depends on a consistent language. We experimented with aligned word embedding models for Italian and English [33] as well, but the results were slightly worse. However, they might still pose a valuable solution for mixed language projects where translation is too costly. We publish the translated versions in our supplementary material [16].

The use cases of eTour, SMOS and eAnci are structured according to commonly-used use case patterns, denoting the use case name, the participating actors, entry and exit conditions as well as the flow of events. eTour's and SMOS' use cases additionally include a short use case description. However, the use cases of the iTrust dataset only consist of event flows.

Table I provides information on how many source and target artifacts are covered by the provided trace links. For iTrust and eAnci the coverage of target and source artifacts respectively is very low. A low coverage of the artifacts indicates that either not all requirements are implemented in the source code, parts of the source code are not described properly, or the gold standard is incomplete. As many approaches consider traceability link recovery as an IR-problem, providing link candidates by searching the most relevant target artifacts for a source artifact, a low coverage leads to many false positives.

Additionally, we calculated precision and F_1-score for a baseline that produces links for each possible pair of source and target artifacts. On most datasets the results are very low. However, on SMOS, which has a low number of artifacts that are highly connected, the baseline F_1-score is 27.4%.

TABLE II
Comparison of Different Feature Selections for FTLR. Variants Using Call Dependencies Are Labeled with CD, Method Comments with MC and Use Case Template Structures with UCT. Best Performing Variants per Project Are Set in Bold Type. p Denotes p-values in a Paired T-Test on the F_1-scores Regarding FTLR as Baseline.

	eTour				iTrust				SMOS				eAnci					
Approach	Pre	Rec	F_1	MAP	Pre	Rec	F_1	MAP	Pre	Rec	F_1	MAP	Pre	Rec	F_1	MAP	∅ F_1	p
FTLR	.287	.455	.352	.330	.151	.297	.200	.227	.417	.140	.209	.398	.215	.125	.158	.142	.230	*base*
+cd	.307	.445	.363	.339	.144	.255	.184	.214	.420	.138	.208	.399	.226	.127	.163	.142	.229	.9683
+mc	.282	.451	.347	.275	.176	.353	**.235**	**.266**	.439	.155	.229	.420	.227	.185	.204	.148	.254	.1180
+uct	.398	.620	.485	**.523**	.151	.297	.200	.227	.426	.277	.336	.418	.306	.194	.237	.146	.315	.0703
+mc +cd	.273	.432	.335	.277	.180	.322	.231	.258	.438	.145	.217	.418	.242	.183	.209	.149	.229	.3019
+uct +cd	.411	.623	**.495**	.516	.144	.255	.184	.214	.439	.277	.340	.421	.314	.194	.240	.146	.315	.1002
+uct +mc	.390	.568	.462	.477	.176	.353	**.235**	**.266**	.443	.297	**.356**	**.442**	.270	.215	.239	**.150**	.323	.0291
+uct +mc +cd	.405	.565	.472	.471	.180	.322	.231	.258	.451	.288	.352	**.442**	.294	.220	**.252**	**.150**	**.327**	.0278

B. Methodology

In all our experiments we use F_1-score and mean average precision (MAP) as the main metrics to evaluate our approach. The former is the preferred metric for classification tasks and is commonly used for traceability link recovery. It is defined as the harmonic mean of precision and recall. In case of traceability link recovery precision reveals how accurate an approach proposes correct trace links. It measures the ratio of correctly proposed links to all proposed links. Recall shows the ability of an approach to propose all correct links. It measures the share of expected trace links that were actually found by an approach. High F_1-scores should be the goal for all automated traceability link recovery approaches [34], as they indicate the approaches' ability to produce the expected results without missing links and producing many false positives.

As current approaches do not come close to the F_1-scores needed for fully automating traceability recovery, most approaches provide a ranked list of candidate trace links per requirement to an expert to ease the manual trace link creation process. To measure the quality of these lists MAP is used. It is defined as the mean average precision across all requirement queries. The average precision (AP) of a query is calculated as follows:

$$AP = \frac{\sum_{r=1}^{|retrieved|}(precision(r) \cdot relevant(r))}{|relevantLinks|}, \quad (5)$$

where $|retrieved|$ is the number of retrieved links for a query, r is the rank (in the list), $precision(r)$ is the *precision* of the list if truncated after rank r, $relevant(r)$ is a binary function that determines whether the link at rank r is valid (1) or not (0), and $|relevantLinks|$ is the total number of links that are relevant for this query according to the gold standard.

To compare our approach with other approaches and to measure whether our approach is able to put relevant links at the top of such a list, we apply our approach without using any of its thresholds. We then generate a ranked list per requirement by using the most similar voter (minimum WMD of the elements) of each class as its representative and sort the list from lowest to highest. Thus, the lists maintain the order of our original approach but additionally include the links we filter out. Note that our approach only considers source code files and some projects include a few non-source code files as gold standard target artifacts. We add them at the end of the list.

In all experiments we either use a publicly available English or Italian pre-trained *fastText* model with 300 dimensions trained on *Wikipedia* and the *CommonCrawl* dataset [29].

VI. Empirical Results

This section presents the results of our empirical studies. We start with analyzing the effect of different feature configurations of FTLR. Then we discuss the impact of the thresholds we use in our approach (see subsection VI-B). Therefore, we systematically vary the thresholds and measure the resulting precision and recall. Finally, we compare the results of FTLR to state-of-the-art approaches.

A. RQ1: Impact of Additional Information Sources

The first research question addresses the impact of the additional information options on the performance of FTLR. To determine the impact, we measured the performance of each feature combination on the four datasets. Table II displays the results. The results show that incorporating all three additional information sources results in the best performing configuration regarding average F_1-score. Additionally, the results of the paired T-test indicate that the improvement in comparison to the basic approach is statistically significant (at the 0.05 level). This is a promising outcome as it supports our claim that all this information can benefit traceability link recovery.

At a closer look at the impacts of the single information sources, it turns out that using use case template structures clearly has the greatest impact. It improves the F_1-scores the most of the three single variants. Only on the iTrust dataset do the results stay the same, as iTrust does not follow a use case template. Using method comments is the second most impactful, as it improves the performance of FTLR on three out of four projects. Only for eTour do method comments consistently worsen the results throughout all configurations. Using call dependencies as the single additional source of information improves the performance on eTour and eAnci but worsens it on iTrust. On SMOS the results stay more or less the same. Depending on the source code structure, a different ratio between the weights of the current method and the call dependencies may improve this result further.

TABLE III
COMPARISON OF THE ORIGINAL FTLR THRESHOLD COMBINATION (ORG) TO THE OPTIMIZED CONFIGURATION PER PROJECT (OPT) ON THE FTLR +UCT+MC+CD VARIANT. ADDITIONALLY, THE USED MAJORITY THRESHOLDS (MAJ.) AND FINAL THRESHOLDS FOR THE OPTIMIZED CONFIGURATION PER PROJECT ARE DISPLAYED.

	Precision		Recall		F_1-Score			Threshold	
Project	ORG	OPT	ORG	OPT	ORG	OPT	Improv.	Maj.	Final
eTour	.405	.456	.565	.516	.472	.484	+.012	.59	.42
iTrust	.180	.231	.322	.273	.231	.250	+.019	.54	.44
SMOS	.451	.370	.288	.455	.352	.408	+.056	.62	.48
eAnci	.294	.240	.220	.282	.252	.259	+.007	.58	.48

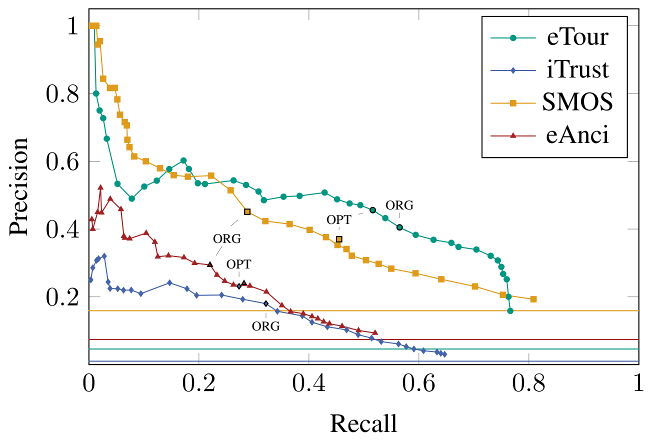


Fig. 3. Precision/recall curves of *FTLR +uct+mc+cd* on the four datasets, if we only vary the final threshold in 0.01 steps between zero and one. ORG and OPT show the respective results obtained with *FTLR's* and the optimized threshold combination. Horizontal lines show the ratio of positive and negative links in the datasets as a baseline.

If we take a look at the combinations of the information sources, combining method comments and use case template structures clearly boosts the performances (compared to being applied individually). Only for eTour and eAnci, *+uct+cd* outperforms the other two-part configurations. On the eTour dataset it achieves the overall highest F_1-score of 49.5%.

The results of the three-part combination indicate that despite the slight decline on iTrust and SMOS the improvement that call dependencies offer on eTour and eAnci results in a better overall performance. However, this improvement is not considerable (0.4 percentage points over *FTLR +uct+mc*) and might be project-dependent. Overall we are confident to state that using use case template structures on artifacts that use these templates and method comments improve performance of our approach in general. Therefore, we consider the combinations *FTLR +uct*, *FTLR +uct+mc*, and *FTLR +uct+mc+cd* as the best performing variants and prefer the last one since it is most versatile across different projects and available information.

B. RQ2: Influence of Threshold Configurations

The second research question investigates whether the fixed thresholds (as defined in section IV) are applicable across different projects. To measure the impact of the thresholds, we determine the ideal thresholds per dataset. Therefore, we systematically altered the majority and final thresholds in 0.01 steps to retrieve the configuration that performs best regarding F_1-score. For the sake of brevity we exemplify the results for our best performing variant *FTLR +uct+mc+cd* only.

Table III shows the performance of the resulting ideal threshold configurations (OPT) in comparison to the thresholds we set for FTLR (ORG). For three out of four projects the optimized configurations improve the F_1-score by less than 2 percentage points. On the eAnci dataset the improvement is even below 0.008. Only on SMOS the optimization results in an improvement of 5.6 percentage points. If we compare the optimized thresholds to FTLR's thresholds (majority = 0.59 and final = 0.44) the differences are small. The respective deviation in the two thresholds are 0.02 and 0.03 on average and remains within $|0.04|$. This implies that our approach of normalized WMD on fine-grained relations enables us to define thresholds that achieve reasonable results across different projects.

Nevertheless, the results on SMOS indicate that a threshold combination tailored to a specific project can improve the results. One way to refine thresholds in future settings, where FTLR is applied to new projects, would be to determine the ideal configuration on a set of projects with gold standard links and use this configuration for new, unseen projects.

To visualize the impact of the final threshold we plot the precision/recall curves achieved when varying the final threshold in 0.01 steps between zero and one in Figure 3. The shapes of the curves indicate that for all threshold combinations the results of FTLR stay well above the respective naive baselines (the horizontal lines). Especially on eTour the achieved difference is apparent. Besides, the curves indicate that the final threshold is more decisive for the performance of FTLR than the majority threshold. For all datasets (except iTrust) the best threshold combination (OPT) is close to or even on the curve when varying only the final threshold. Note that each of the curves ends at a certain recall level. FTLR discards some link candidates earlier because of the majority threshold. Therefore, our approach is unable to attain full recall.

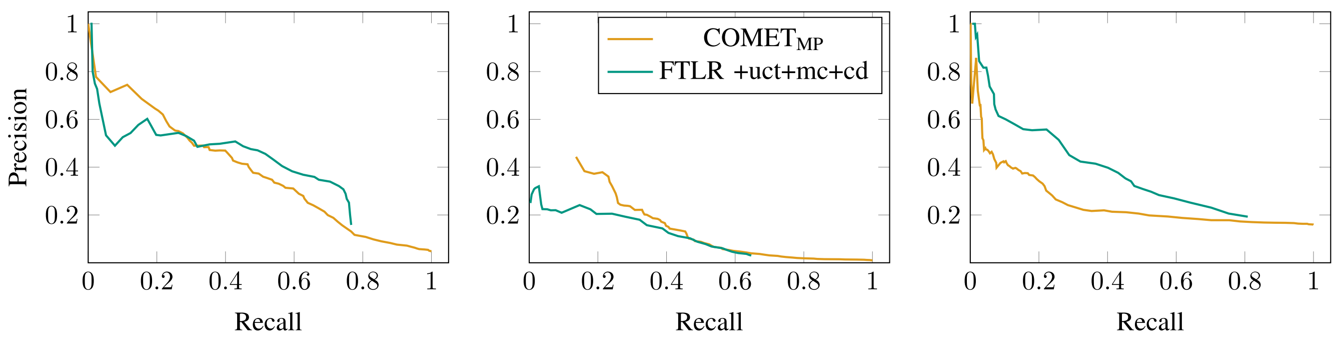
C. RQ3: Comparison to State-of-the-Art

To answer RQ3 we use five recent approaches to compare our results to. We limit the comparison to these approaches as they are either closely related or we had access to their detailed results. Unfortunately, most other related work do not provide data or tools/code. Therefore we publish our translated datasets and code [16]. We use all datasets introduced in subsection V-A except for eAnci, as we had no access to detailed results of unsupervised approaches on eAnci to compare to. We choose the approaches WQI [11] and S2Trace [14], as they also use embeddings for traceability link recovery. Additionally, we compare our results to the first stage results of COMET [9], as its unsupervised approach performs well on unseen projects[4]. We use only the first stage because the further stages of COMET include either developer feedback (not automatic) or transitive

[4]Note that we compare to the version of COMET using Maximum a Posteriori (MP) estimation as it had the best F_1-score on the datasets.

TABLE IV
Comparison of FTLR to State-of-the-art Approaches. Brackets Indicate Results on the Full iTrust Solution Set (Including JSP). Approaches Are Distinguished by Whether They Are Supervised (sup) or Unsupervised (unsup).

		eTour				iTrust				SMOS			
	Approach	Pre	Rec	F_1	MAP	Pre	Rec	F_1	MAP	Pre	Rec	F_1	MAP
U	S2Trace	.101	.364	.158		(.196)	(.417)	(.267)					
N	COMET_{MP}	.412	.464	.437	.467	.361	.231	.282	.252	.166	.816	.276	.293
S	**FTLR +uct**	.398	.620	.485	.523	.151	.297	.200	.227	.426	.277	.336	.418
U	**FTLR +uct +mc**	.390	.568	.462	.477	.176	.353	.235	.266	.443	.297	.356	.442
P	**FTLR +uct +mc +cd**	.405	.565	.472	.471	.180	.322	.231	.258	.451	.288	.352	.442
S	WQI	.088	.415	.145		(.198)	(.322)	(.245)					
U	$\text{ALCATRAL}_{10\%}$	.425	.427	.425		(.504)	(.228)	(.309)		.513	.444	.476	
P	TRAIL	.572	.650	.608		(.568)	(.658)	(.609)		.871	.735	.797	



(a) eTour (b) iTrust (c) SMOS

Fig. 4. Precision/recall curves of *FTLR +uct+mc+cd* in comparison to COMET_{MP}, if we only vary the final threshold in 0.01 steps between zero and one.

links, which are not available in many projects. For COMET, we had access to the resulting ranked list and thus were able to additionally calculate the best F_1-score and MAP (instead of average precision only as provided by the authors). Furthermore, we calculate their results on the reduced iTrust dataset (without JSP files) and thus make their results comparable to ours.

Besides WQI, that uses learning-to-rank, we add TRAIL [25] and ALCATRAL [12] as supervised approaches to the comparison. The results of TRAIL are calculated by performing 50 random split runs with 90% of the gold standard as training data each. ALCATRAL uses active learning to improve TRAIL's performance with less training data. We use the version of ALCATRAL that uses only 10% of the gold standard for training, as it is the closest to an unsupervised setting. Note that a comparison of unsupervised to supervised approaches can only put the unsupervised results into context but does not pose a fair comparison. Supervised approaches are only applicable in settings where gold standard trace links are already available and require a significant amount of these links. In contrast, unsupervised approaches are applicable to any project without investing resources into creating initial links.

Table IV provides the results of FTLR in comparison to state-of-the-art approaches. The results indicate that FTLR outperforms unsupervised approaches in F_1-score on two out of three projects. In comparison to COMET, *FTLR +uct+mc+cd* achieves an improvement in F_1-score of more than three percentage points on eTour and more than seven on SMOS. On eTour this means that FTLR identifies 31 more correct links than COMET. This is a promising result since FTLR recovers 174 of the 308 expected links out of 6728 possible links (with only 256 false positives). On SMOS the achieved MAP of FTLR is more than 14 percentage points higher than COMET's, whereas on eTour they are similar. For *FTLR +uct*, the results show an even wider gap in F_1-score (+0.052) and MAP (+0.056) on the eTour dataset. The higher MAP indicates that our approach places the relevant trace link candidates more closely to the top of the candidate list than COMET. Thus, the fine-grained word embedding-based similarities are able to identify relatedness between the artifacts better. For iTrust the MAP and recall of FTLR with method comments are higher than COMET's. However, FTLR lacks precision. One reason might be that the iTrust dataset only includes the flow of events of its use case descriptions. Thus, no information on the use case name or description can be used. Additionally, many classes in iTrust only contain private or protected methods. For these classes our approach only uses the class name for comparison (which does not utilize the full potential of our approach). If the class name is similar to many requirements, FTLR struggles with identifying the correct links without additional information. Additionally, if many classes inherit from the same superclass/interface and only provide the same methods, FTLR can solely distinguish them by their class names. For SMOS and eAnci, which have many links to servlet classes that only contain protected methods, this might explain the rather low recall. If the class names are not semantically similar to the requirements they should be linked to, FTLR does not have sufficient information to identify the links. In general, FTLR has difficulties with classes that should be linked to many requirements. This is due to the decision to only use the requirement(s) with the most votes as link candidates. It is a trade off between precision and recall.

Figure 4 shows the precision/recall curves of the best performing FTLR variant in comparison to COMET's. Again we altered the final threshold in 0.01 steps. The curves support the insights gained from Table IV. On the SMOS dataset FTLR performs better than COMET on all recall levels. For eTour our approach outperforms COMET between 0.4 and 0.75 with a more stable precision (lower decline). However, the lack in precision on the iTrust dataset also shows in the curves.

Finally, we compare our results to supervised approaches. As the two settings differ regarding prerequisites and applicability, this only provides insights into how close FTLR's performance is to state-of-the-art approaches that require gold standard links for training. However, the results of this comparison are promising. On the eTour dataset all FTLR variants outperform $\text{ALCATRAL}_{10\%}$ and *FTLR +uct* achieves only 12.3 percentage points less than TRAIL. As TRAIL uses 90% of the gold standard for training, the manual effort to achieve this result is enormous. An explanation for the difference might be that supervised approaches can infer project-specific relations from the training data which are not represented in our word embedding model. In comparison to the embedding-based approaches WQI and S2Trace we can state that on the eTour dataset FTLR is superior. It outperforms both approaches by over 30 percentage points in F_1-score. We attribute this to our different approach at using word embeddings. FTLR uses a pre-trained *fastText* model instead of training a model on the datasets itself, calculates similarities with WMD instead of cosine similarity, and uses fine-grained relations.

VII. Threats to Validity

In this section, we discuss potential threats to validity of our research and experimental design.

External Validity: The first and probably most major threat to validity of our work concerns external validity. We evaluate our approach on a limited number of projects stemming from academic projects. Thus, it is possible that our results are not representative and might not generalize to other projects. This threat is shared by automated traceability link recovery approaches [9], [10]. However, the chosen projects are widely used and accepted in the research community, are of different sizes, and cover different domains.

Additionally, all datasets contain more or less well-written requirements. Thus, our results might not be transferable to projects that include requirements of poor quality. Especially, the utilization of use case templates might not be applicable. However, our word embedding-based approach itself does not require syntactically correct sentences.

FTLR requires the word embedding model to cover the semantic relations needed to identify the trace links. The results indicate that the used model enables FTLR to discern relations other approaches can not identify. Again, this might not be the case for other projects and domains.

Internal Validity: A threat might be that one of the authors translated the identifiers in two projects. As the translation was performed with the goal of traceability link recovery in mind, the translation may suffer from experimenter bias. We mitigate this risk by publishing the translated datasets [16].

Additionally, we compare our results to results reported in publications without reproducing their results. As COMET's results were released online, we were at least able to adapt their results to the artifacts we used. However, for eTour and SMOS we translated parts of the datasets and thus compare FTLR's results on the translated datasets to results of recent approaches on the original (non-translated) datasets. This poses a potential threat to the validity of the comparison. However, we believe the results to be comparable because COMET, ALCATRAL, and TRAIL use information retrieval techniques that solely rely on the present wording (replicating their syntactic relations) and do not use language sources (such as pre-trained word embeddings) that rely on a consistent language usage. Same holds true for the approaches that use embeddings, as their embedding models are trained on the datasets themselves.

VIII. Conclusion

We presented the approach Fine-grained Traceability Link Recovery (FTLR) that utilizes fine-grained requirements-to-code relations for traceability link recovery. It uses word embeddings and the Word Mover's Distance to compute similarities between artifact elements and aggregates the results with majority votes to recover artifact-level trace links.

We performed a three-part empirical study on the projects eTour, iTrust, SMOS and eAnci commonly used in the community to examine FTLR's performance. We have shown that using method call dependencies, method comments, and use case template structures as additional resources significantly improves FTLR's performance on the four test datasets. By comparing the results of our fixed thresholds to the optimal threshold configuration per project, we show that the thresholds we define provide reasonable performance across different projects. Compared against recent unsupervised traceability link recovery approaches, FTLR performs better regarding F_1-score and MAP. It attains F_1-scores of up to 49.5% on tasks such as retrieving the 308 relevant links out of all 6728 possible end-to-end links in eTour. However, an average F_1-score of 32.7% is still far from a performance required to replace manual trace link generation and further work is needed.

We plan to investigate possibilities to further enrich the knowledge about the artifacts. For example, we will incorporate underlying concepts from external knowledge bases to bridge the semantic gap between the artifacts. Using language models, such as BERT [35] or CodeBERT [36], could present another approach to close the gap and improve the representation of the artifacts. Additionally, we will explore different opportunities to retrieve a generally applicable threshold combination.

References

[1] J. Cleland-Huang, O. Gotel, A. Zisman *et al.*, *Software and systems traceability*. Springer, 2012, vol. 2, no. 3.

[2] L. Rierson, *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press, Jan. 2013.

[3] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, Oct. 2002.

[4] A. Marcus and J. I. Maletic, "Recovering Documentation-to-source-code Traceability Links Using Latent Semantic Indexing," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 125–135.

[5] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 133–142.

[6] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang, "Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 378–388.

[7] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.*, Sep. 2003, pp. 138–147.

[8] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: A study of term-based enhancement methods," *Empir Software Eng*, vol. 15, no. 2, pp. 119–146, Apr. 2010.

[9] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, and J. Johnson, "Improving the effectiveness of traceability link recovery using hierarchical bayesian networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 873–885.

[10] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically Enhanced Software Traceability Using Deep Learning Techniques," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 3–14.

[11] T. Zhao, Q. Cao, and Q. Sun, "An Improved Approach to Traceability Recovery Based on Word Embeddings," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2017, pp. 81–89.

[12] C. Mills, J. Escobar-Avila, A. Bhattacharya, G. Kondyukov, S. Chakraborty, and S. Haiduc, "Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2019, pp. 103–113.

[13] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, May 2010, pp. 95–104.

[14] L. Chen, D. Wang, J. Wang, and Q. Wang, "Enhancing Unsupervised Requirements Traceability with Sequential Semantics," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2019, pp. 23–30.

[15] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[16] T. Hey, F. Chen, S. Weigelt, and W. F. Tichy, "Supplementary Material of "Improving Traceability Link Recovery Using Fine-grained Requirements-to- Code Relations"," Jul. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5119858

[17] P. Rempel, P. Mäder, and T. Kuschke, "Towards feature-aware retrieval of refinement traces," in *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, May 2013, pp. 100–104.

[18] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods," *IEEE Trans. Softw. Eng.*, vol. 32, no. 1, pp. 4–19, Jan. 2006.

[19] N. Assawamekin, T. Sunetnanta, and C. Pluempitiwiriyawej, "Ontology-based multiperspective requirements traceability framework," *Knowl Inf Syst*, vol. 25, no. 3, pp. 493–522, Dec. 2010.

[20] S. Hayashi, T. Yoshikawa, and M. Saeki, "Sentence-to-Code Traceability Recovery with Domain Ontologies," in *2010 Asia Pacific Software Engineering Conference*, Nov. 2010, pp. 385–394.

[21] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "When and How Using Structural Information to Improve IR-Based Traceability Recovery," in *2013 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 199–208.

[22] H. Kuang, P. Mäder, H. Hu, A. Ghabi, L. Huang, J. Lü, and A. Egyed, "Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code?" *J. Softw. Evol. Process*, vol. 27, no. 11, pp. 838–866, Nov. 2015.

[23] J. M. Florez, "Automated Fine-Grained Requirements-to-Code Traceability Link Recovery," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, May 2019, pp. 222–225.

[24] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requirements Eng*, vol. 20, no. 3, pp. 281–300, Sep. 2015.

[25] C. Mills, J. Escobar-Avila, and S. Haiduc, "Automatic Traceability Maintenance via Machine Learning Classification," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2018, pp. 369–380.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Jan. 2013.

[27] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, Dec. 2017.

[28] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From Word Embeddings To Document Distances," in *International Conference on Machine Learning*. PMLR, Jun. 2015, pp. 957–966.

[29] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[30] "Natural Language Toolkit — NLTK 3.6.2 documentation," https://www.nltk.org/, (accessed 2021-06-23).

[31] "spaCy · Industrial-strength Natural Language Processing in Python," https://spacy.io/, (accessed 2021-06-23).

[32] "Center of Excellence for Software & Systems Traceability (CoEST) - Datasets," http://sarec.nd.edu/coest/datasets.html, (accessed 2021-06-23).

[33] A. Joulin, P. Bojanowski, T. Mikolov, H. Jégou, and E. Grave, "Loss in Translation: Learning Bilingual Word Mapping with a Retrieval Criterion," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2979–2984.

[34] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, "The Grand Challenge of Traceability (v1.0)," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer, 2012, pp. 343–409.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[36] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547.