Karlsruhe Institute of Technology

Assessing Word Similarity Metrics For Traceability Link Recovery

Bachelor's Thesis of

Kevin Werber

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr.-Ing. Anne Koziolek
Second reviewer:	Prof. Dr. Ralf Reussner
Advisor:	Jan Keim, M.Sc.
Second advisor:	Tobias Hey, M.Sc.

27. January 2022 – 27. May 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

The software development process usually involves different artifacts that each describe different parts of the whole software system. Traceability Link Recovery is a technique that aids the development process by establishing relationships between related parts from different artifacts. Artifacts that are expressed in natural language are more difficult for machines to understand and therefore pose a challenge to this link recovery process. A common approach to link elements from different artifacts is to identify similar words using word similarity measures. ArDoCo is a tool that uses word similarity measures to recover trace links between natural language software architecture documentation and formal architectural models. This thesis assesses the effect of different word similarity measures on ArDoCo. The measures are evaluated using multiple case studies. Precision, recall, and encountered challenges for the different measures are reported as part of the evaluation.

Zusammenfassung

Der Softwareentwicklungsprozess involviert oft verschiedene Artefakte, welche jeweils verschiedene Aspekte eines Softwaresystems beschreiben. Traceability Link Recovery ist ein Verfahren, das diesen Entwicklungsprozess unterstützt, indem es verwandte Teile aus verschiedenen Artefakten verbindet. Artefakte, die in natürlicher Sprache ausgedrückt werden, sind schwierig für Maschinen zu verstehen und stellen damit eine besondere Herausforderung für die Traceability Link Recovery dar. Hierfür werden für gewöhnlich Wortähnlichkeitsmetriken eingesetzt, um unterschiedliche Wörter mit gleicher Bedeutung als Synonyme zu identifizieren. ArDoCo ist eine Software, die Wortähnlichkeitsmetriken zum Wiederherstellen von Trace Links zwischen textueller Softwarearchitekturdokumentation und formalen Architekturmodellen einsetzt. Diese Arbeit befasst sich mit dem Einfluss verschiedener Wortähnlichkeitsmetriken auf ArDoCo. Die Wortähnlichkeitsmetriken werden mit mehreren Fallstudien evaluiert. Dazu werden die Metriken Präzision und Sensitivität als auch besondere Herausforderungen der einzelnen Wortähnlichkeitsmetriken als Teil der Evaluation präsentiert.

Contents

Abstract i

Zusammenfassung iii

1 Introduction 1

2 Foundations 3
- 2.1 Traceability Link Recovery 3
- 2.2 Word Similarity 5
- 2.3 ArDoCo 6

3 Related Work 9
- 3.1 Relevant Word Similarity Measures 9
  - 3.1.1 Lexical similarity measures 9
  - 3.1.2 Knowledge-based similarity measures 10
  - 3.1.3 Corpus-based similarity measures 11
  - 3.1.4 Hybrid measures 12
- 3.2 Related Traceability Link Recovery Work 12

4 Implementation 15
- 4.1 Choosing Word Similarity Metrics 15
- 4.2 Architecture 17

5 Evaluation 19
- 5.1 Methodology 19
- 5.2 Case studies 20
- 5.3 Baselines 20
- 5.4 Results 21
- 5.5 Discussing the results 30
  - 5.5.1 Overview 30
  - 5.5.2 Accuracy 31
  - 5.5.3 Reliability 31
  - 5.5.4 Validity 32
  - 5.5.5 Scope 32

6 Conclusion and Future Work 33

Bibliography 35

List of Figures

1.1 Example of a textual SAD (left) and an architectural model (right) 1
1.2 Example trace links between textual SAD and an architectural model . . 2

2.1 Trace links between textual SAD and an architectural model 4
2.2 Word Similarity Measure Categories from Pradhan et al. [7] 5
2.3 ArDoCo pipeline 7

4.1 The main classes of ArDoCo's WSM architecture 17

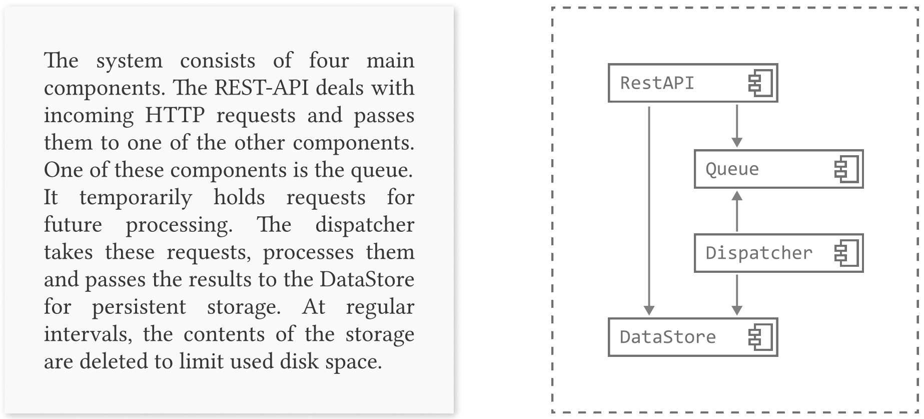
5.1 Jaro Winkler evaluation graph 22
5.2 The precision and recall pairs of all evaluated Levenshtein configurations 23
5.3 Comparing different values of *n* for Kondrak's N-gram WSM 24
5.4 N-gram evaluation graphs for $n = 3$ 24
5.5 SEWordSim evaluation graphs 25
5.6 Comparison of different fastText models 26
5.7 fastText cc.en.300 evaluation graphs 26
5.8 Comparison of WordNet algorithms 27
5.9 Comparison of the GloVe vector embeddings 28
5.10 GloVe evaluation graphs using the 300 dimensional `WIGI` embeddings . . 28
5.11 Nasari evaluation graph 29
5.12 Comparison of all WSMs 30
5.13 Extended Jaro Winkler Evaluation Graph 31

List of Tables

5.1 General statistics about the used case studies 20
5.2 Evaluation results for the first baseline 20
5.3 Evaluation results for the second baseline 21

1 Introduction

When developing software, different views are usually involved that each describe a part of the desired software system. Different views emphasize different aspects of the software. For example, an architectural model of a system describes how different software components are connected and the source code provides implementation details for each component. Additionally, some views are more structured than others. While any element in a formal model follows a specification defined for that model, textual documentation is usually less strict and subject to fewer rules. Figure 1.1 shows an example of a textual software architecture documentation (SAD) on the left and an architectural model on the right.



The system consists of four main components. The REST-API deals with incoming HTTP requests and passes them to one of the other components. One of these components is the queue. It temporarily holds requests for future processing. The dispatcher takes these requests, processes them and passes the results to the DataStore for persistent storage. At regular intervals, the contents of the storage are deleted to limit used disk space.

Figure 1.1: Example of a textual SAD (left) and an architectural model (right)

A problem that can occur when different views cover the same parts of a system is that inconsistencies might occur between these views. Therefore, special care needs to be taken, so that no inconsistencies occur whenever a view is changed. The research project ArDoCo [1] was created to assist in identifying such inconsistencies. It accomplishes this by utilizing Traceability Link Recovery (TLR). The TLR process in ArDoCo creates trace links between elements from an architectural model and sentences from the textual SAD when both refer to the same software component. By automatically linking these parts from different views, circumstances where inconsistencies might occur are found and possibly automatically recognized as inconsistent (cf. Keim et al. 2021 [2]). Figure 1.2 shows possible trace links from the previous example. One such trace link connects the term *REST-API* from the textual SAD to the `RestAPI` component in the model. Since they both refer to the same component, a link is established.

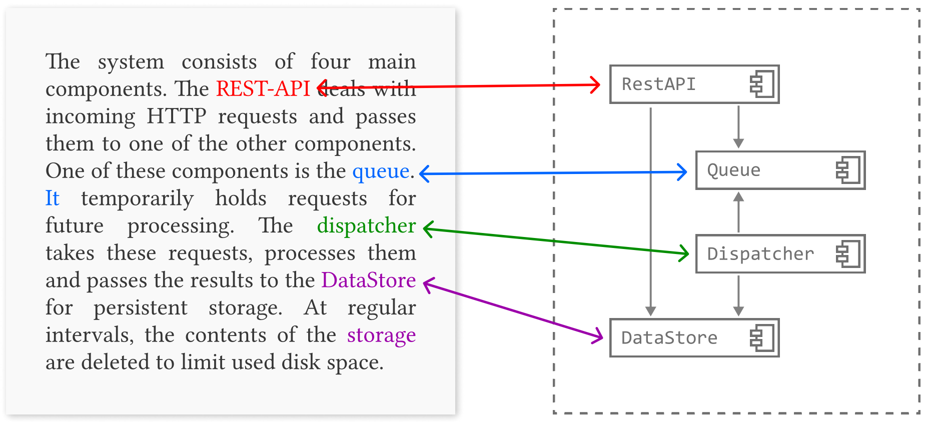

The system consists of four main components. The REST-API deals with incoming HTTP requests and passes them to one of the other components. One of these components is the queue. It temporarily holds requests for future processing. The dispatcher takes these requests, processes them and passes the results to the DataStore for persistent storage. At regular intervals, the contents of the storage are deleted to limit used disk space.

Figure 1.2: Example trace links between textual SAD and an architectural model

In order for ArDoCo to understand that two terms refer to the same thing, it utilizes word similarity measures (WSMs). It uses these WSMs by letting them compute a sense of similarity between the names of architectural models and terms occuring in the textual SAD. If the computed similarity passes a certain threshold, the elements are linked. As is the nature of written text, different terms in the textual SAD might refer to the same thing. Using the previous example, both the terms *DataStore* and *storage* refer to the same software component. But since the actual name of the component is `DataStore`, ArDoCo would fail to link this component to the term *storage* without the usage of WSMs. By utlizing WSMs, ArDoCo can determine that both terms are similar enough that they most likely refer to the same component. Another problem in textual SAD is the occurence of coreferences. In Figure 1.2, the link between the word “It” and the `Queue` component is missed. However, since this problem cannot be solved with the usage of word similarity measures, it is not relevant for this thesis.

There are many different WSMs in the field of word similarity and their potential impact on TLR performance between natural language SAD and formal architectural models is yet unexplored. To the best of my knowledge, no existing work in the literature focuses on the effects of WSMs on TLR. Instead, WSMs are often only mentioned in passing as an extension to the actual TLR approaches. This thesis aims to investigate the effect of WSMs by attempting to answer a series of research questions:

- How do state-of-the-art word similarity measures affect the performance of TLR between natural language SAD and formal architectural models?
- What kinds of problems are encountered when utilizing WSMs for TLR?
- Which kinds of WSMs are particulary effective or ineffective when used for TLR?

To answer these questions, various WSMs are implemented in ArDoCo and their performance impact is evaluated using multiple case studies. The rest of this thesis is structured as follows: The upcoming chapter establishes the foundations for this thesis. Chapter 3 presents existing work that is related to this thesis. Chapter 4 describes the implementation work for this thesis and Chapter 5 presents the evaluation results. Lastly, Chapter 6 summarizes and concludes the results of this thesis.

2 Foundations

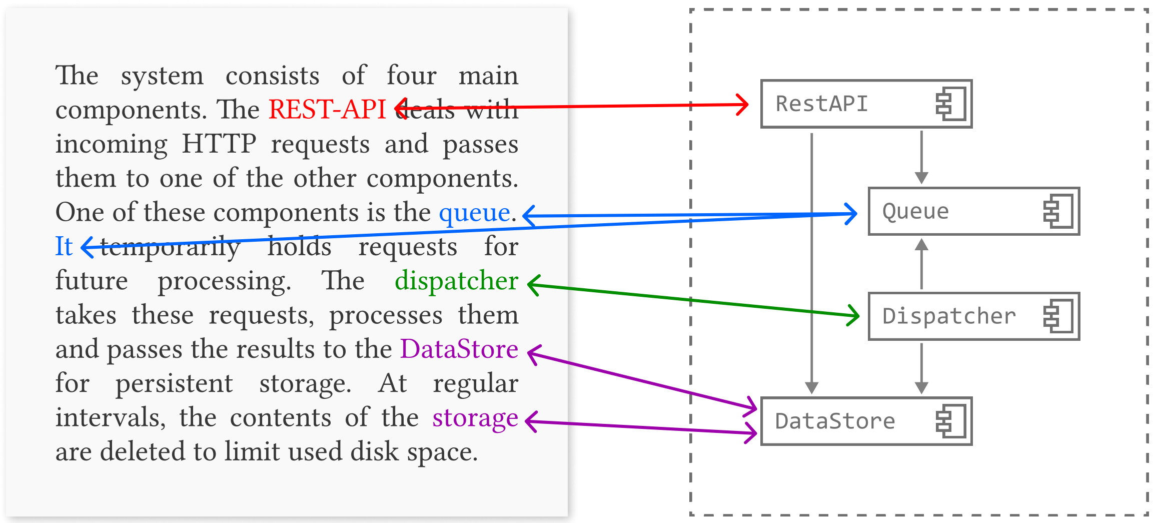
The work of this thesis is based on the field of traceability link recovery and the field of word similarity. The foundations for these two research fields are explained in the upcoming sections 2.1 and 2.2. Section 2.3 provides an overview of ArDoCo and its relevant parts.

2.1 Traceability Link Recovery

Traceability is concerned with the "degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [3]". The idea behind Traceability Link Recovery (TLR) is to recover links between related elements from certain software artifacts. A TLR process can be thought of as a function that, given two or more software artifacts, provides a set of trace links between these artifacts. Each trace link connects an element from one artifact to an element from another artifact and indicates a relationship between these elements. The trace links can then be used to assist the software development process in different ways:

- Linked elements can provide additional information and reasoning about the design of an inspected element. For example, the requirements linked to a part of the source code explain that code's purpose.
- The established links can be used to identify places in software artifacts where inconsistencies might occur. Developers can therefore check these places to ensure that no inconsistencies exist.
- The absence of links to a specific artifact can indicate that the artifact is incomplete.
- After changing a certain part of an artifact, the developer can quickly check all elements from other artifacts that are linked to the changed part to see if they have introduced inconsistencies.

Cleland-Huang et al. [4] provide an introduction and overview into the field of TLR.



The system consists of four main components. The REST-API deals with incoming HTTP requests and passes them to one of the other components. One of these components is the queue. It temporarily holds requests for future processing. The dispatcher takes these requests, processes them and passes the results to the DataStore for persistent storage. At regular intervals, the contents of the storage are deleted to limit used disk space.

Figure 2.1: Trace links between textual SAD and an architectural model

Figure 2.1 serves as an example of TLR between two different artifacts. In this example, the textual SAD would be incomplete if a developer would add another component to the model on the right but forget to add the respective description of that component to the artifact on the left. This problem would then be noticed since the model component has no outgoing trace links.

There are manual, semi-automated, and fully automated approaches for TLR. Semi-automated approaches usually work by suggesting possible links to a human who then either approves or declines the suggestion. Human involvement is always slow and costly. This effect is exaggerated when dealing with very big or frequently changing artifacts.

A different way to classify TLR approaches is by looking at the involved artifacts. Commonly involved artifacts are source code, software requirements, formal architectural models, use cases, test cases, bug reports and user documentation [5]. A general problem of TLR is that different kinds of artifacts exist at different levels of granularity, are structured differently and contain different information. While source code reveals the inner workings of a software system, requirements exist at a higher layer of abstraction and describe a more general purpose of a system. Additionally, artifacts written in natural language are more difficult for automated processes to understand, compared to more structured artifacts that follow strict specifications.

2.2 Word Similarity

The field of natural language processing has developed many different methods to measure the similarity of two words. Navigli and Martelli [6] provide a formal definition of a word similarity measure (WSM) that applies to all methods:

$$sim : I \times I \longrightarrow \mathbb{R}$$

A similarity function *sim* is defined, where I is the set of all relevant words. The output of a similarity function usually lies between 0 and 1 or between -1 and 1. The larger the output number, the more similar the input words. Another way to formalize word similarity is to instead determine how different words are, by defining a distance function:

$$dist : I \times I \longrightarrow \mathbb{R}_+$$

In this case, smaller output numbers imply that words are more similar and greater numbers imply that the words are more different. Both definitions can be applied to any WSM since one definition can always be transformed into the other.

There are many different ways of approaching the word similarity problem. Pradhan et al. [7] classifies different WSMs into the categories seen in Figure 2.2.

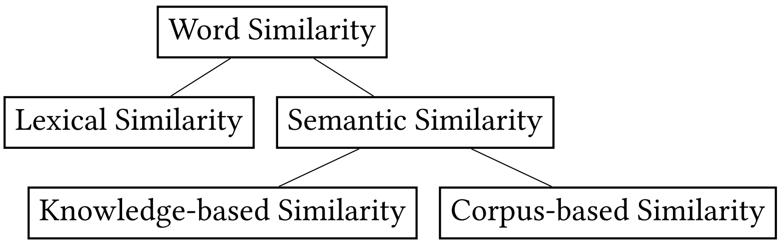


Figure 2.2: Word Similarity Measure Categories from Pradhan et al. [7]

Lexical similarity measures determine similarity by looking at the characters of the words. These kinds of WSMs are good at identifying words that *look* similar and especially at identifying typos. However, these measures do not work well in cases where the words do not look similar, despite having the same meaning. For example, the words *answer* and *reply* are synonymous but look very different. Since the only used information comes from the characters themselves, lexical measures have no way of knowing that *answer* and *reply* are similar.

Semantic similarity measures determine similarity by attempting to gauge the actual meaning of a word. The two main types of semantic similarity measures are knowledge-based and corpus-based measures.

As defined by Pradhan et al. [7], knowledge-based similarity measures utilize information gained from semantic networks. These networks establish relations between word senses and provide ways for machines to understand the meaning of words. An example for one such relation would be the so called *is-a* relation which exists between two words w_1 and w_2, if w_1 is a special instance of w_2, like in the case of *car* and *vehicle*. A problem of knowledge-based approaches is that the construction of such semantic networks is difficult and often requires manual labor.

Corpus-based similarity measures try to utilize information gathered from huge corpora. The idea behind this is that the meaning of a word can be derived just by looking at the contexts of where the word is used. Approaches of this kind analyze corpora and convert words from the corpora into numerical vectors that can be used to compare the words. The procedure that converts words into such vectors is called a word embedding. A common word embedding approach is word co-occurence where vectors are constructed by analyzing which words occur around a word. The idea here is that two words are deemed similar if their surrounding words are similar. Once vector representations for words are constructed, their similarity can be calculated with vector similarity measures. One such vector similarity measure is cosine similarity which treats the cosine of the angle between vectors as a measure of similarity.

The benefit of the corpus-based approach compared to knowledge-based similarity is that accessing and analyzing huge corpora is easier than manually constructing semantic networks. This is because a corpus can be just a body of natural language text. However, word senses that rarely occur in written text are at risk of not being sufficiently understood. For example, to learn the meaning of the word "cloud" in the context of software engineering, the WSM must analyze a sufficiently large amount of corpora from the software engineering domain. This means that problems arise when corpus-based WSMs have to compute word similarity in very specific domains where gathering huge amounts of written text related to that domain is infeasable. In cases where huge amounts of relevant corpora are available, the greater amount of words along with their contexts means that corpus-based WSMs are able to understand more unique words than knowledge-based approaches.

Another challenge that all WSMs deal with is the problem of polysemy. The fact that a single word can have multiple meanings necessitates that, in order to truly gauge the real meaning of a word, the context of where that word occurs needs to be taken into account. Lexical measures have no way of utilizing contextual information since they only look at the characters of a given word. Knowledge-based measures can utilize context to find the appropriate word sense in a semantic network and then use the relationships within the network to further compute similarity. Corpus-based measures must provide ways to construct vectors using words combined with their respective context.

2.3 ArDoCo

ArDoCo [1] is a research project started by the research group Modelling for Continuous Software Engineering at the Karlsruhe Institute for Technology. The goal of this project is to provide consistency analyses between textual SAD written in natural language and formal architectural models. The core framework of ArDoCo [8] is an open-source project written in Java. This framework implements the TLR process described by Keim et al. [2]. To assess the performance impact of WSMs on this TLR process, various WSMs from all categories were implemented and evaluated in ArDoCo. The TLR approach for ArDoCo aims to automatically create trace links between sentences from textual SAD and elements from architectural models. A link is created between a sentence and an element of a model, when a word from the sentence and the model element refer to the same software component.

ArDoCo's consistency analysis is realized through a multi-stage process where each stage uses the results of the previous stage to gain new information. Figure 2.3 shows all stages and how they are connected. Word similarity measures are used at multiple stages in this process. The extraction stage is tasked with locating words in the SAD that could possibly represent names or types of model elements. Words that refer to the same name or type are grouped together as so-called noun mappings. WSMs are used for this by grouping together words that are deemed similar enough. The recommendation stage attempts to reconstruct the model using information from the SAD and metamodel. Model elements that are constructed during this stage are called recommended instances. Before constructing a new recommended instance, WSMs are used to find out if any similar recommended instances already exist. The connection stage is where trace links are created. During this stage, WSMs are used to find noun mappings and words that are similar to model elements. Additionally, similarity comparisons are made between model elements and recommended instances.

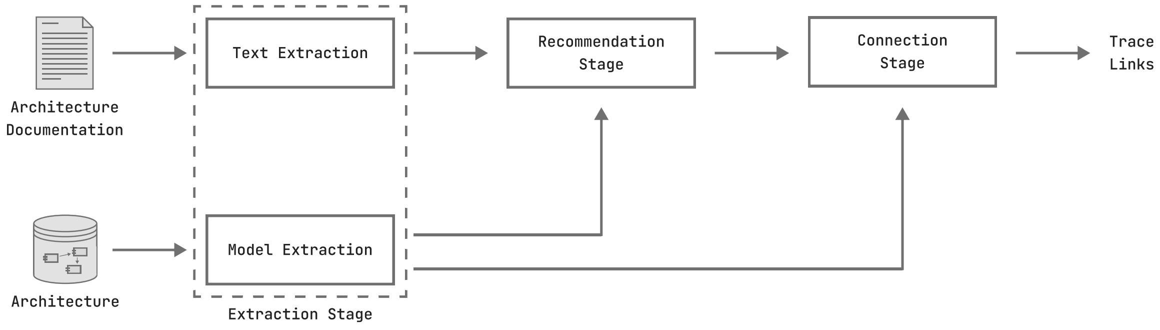


Figure 2.3: ArDoCo pipeline

3 Related Work

This chapter covers existing work that is related to this thesis. As for explicitly evaluating multiple WSMs on TLR, no similar work has been found in the literature. Section 3.1 presents relevant word similarity measures and Section 3.2 covers work related to the field of TLR.

3.1 Relevant Word Similarity Measures

Several works from the literature provide an overview over word similarity measures. Navigli and Martelli [6] provide an introduction and overview into the problem of word similarity. Pradhan et al. [7] also provide an overview over existing WSMs. Chandrasekaran and Mago provide "a comprehensive view of existing systems in place for new researchers to experiment and develop innovative ideas to address the issue of semantic similarity [9]".

The following subsections cover the most popular and relevant measures for each WSM category.

3.1.1 Lexical similarity measures

This subsection presents measures that rely on lexical similarity for word comparisons.

N-Gram by Kondrak [10] is a family of lexical similarity measures. For each member of this family, the author provides a distance-based and a similarity-based variant. The idea behind this approach is identifying substrings of a specific length n (so-called n-grams) for each input string and comparing these n-grams with each other to calculate similarity or distance. Special members of this family are the Levenshtein distance and the longest common subsequence measure.

Levenshtein distance [11] is a distance function that, given two input words, calculates how many characters have to be added, moved or removed from the first word to turn it into the second word. The more modifications are necessary, the less similar the two words are. This measure is also sometimes called *EDIT-Distance* in the literature.

Longest common subsequence (LCS) is a similarity function that finds the longest subsequence that occurs in both of the input words and treats the length of that subsequence as the similarity score. In this context, a subsequence of a string X is a series of characters $c_1c_2 \dots c_k$ that occur in X in that exact order. The characters however do not need to be right next to each other. For example, one subsequence of X = "s**im**il**a**ri**t**y" is "imat". The LCS of X = "si**mil**arit**y**" and Y = "fa**mily**" is "mily". The longer the LCS of two words X and Y, the more similar the words are.

Jaro Similarity [12, 13] is a similarity function that counts how many matching characters both words share. Two characters match if they are the same characters from different words and their positions differ by less than half the length of the shorter word. This prevents mistakes where characters are swapped from negatively affecting similarity. This measure also takes into account the order of matching characters by decreasing similarity when the order of matching characters in one word is different in the other word. For example, the words `"socket"` and `"aspect"` have four matching characters. Since the matching characters occur in different orders, the similarity is slightly decreased.

Jaro Winkler is an extension of the Jaro measure [14]. It provides an additional increase in similarity when both words share a common prefix.

3.1.2 Knowledge-based similarity measures

This subsection covers measures that use semantic networks for similarity comparison.

WordNet [15] is one of the most popular semantic networks. In this network, synonym sets (synsets) of specific word senses are nodes in a graph and edges between nodes represent semantic relationships. One such semantic relationship in this graph is the hyponymy relationship. It conveys that one synset is a *type-of* another synset, like in the case of “vehicle” and “bicycle”. With this semantic relationship, the WordNet graph becomes a tree structure where the root node is the “entity” synset. Aside from edges, WordNet also provides a textual definition (gloss) for each synset. Meng et al. [16] provide a review on several measures that work with WordNet and put them into different categories. Some of these measures will be mentioned in the following paragraphs. Banu et al. [17] and Goyal [18] performed an evaluation on the most common WordNet measures.

BabelNet is a multilingual semantic network that is automatically constructed by combining several existing resources like WordNet and Wikipedia. Each concept in BabelNet has a reference to at least one if not more counterparts from other resources. For example, the BabelNet concept “car” (bn:00007309n) is linked to the WordNet synset `car%1:06:00::` and to the Wikipedia page `wikipedia.org/wiki/Car`. Each BabelNet concept can also have semantic relations to other concepts in the same way that WordNet synsets have semantic relationships. All of these concepts and relationships are automatically imported from multiple different resources. To avoid having duplicate concepts, BabelNet maps concepts from different sources together, like in the case of the previously mentioned “car” concept.

The following measures all utilize information from the WordNet graph. Many of these measures actually calculate the similarity between synsets and not between words. A common way to turn these measures into WSMs is to calculate the maximum similarity

$$\max_{s_1 \in S(w_1), s_2 \in S(w_2)} sim(s_1, s_2)$$

where $S(w_i)$ is the set of all synsets that contain the word w_i.

Path [19] is one of the earliest WordNet measures. It calculates the length of the shortest path from one synset to another synset in the WordNet graph. The idea being, that synsets are less similar the farther away they are from each other.

Leacock & Chodorow [20] is an extension of the Path algorithm. It also calculates the shortest distance between synsets but normalizes this distance by the maximum depth of the WordNet hierarchy. The depth of a node in this context refers to the distance between the node and the root “entity” synset using the hyponymy edges.

Wu & Palmer [21] utilizes depth information and the least common subsumer (LCS) of both synsets. The LCS of two synsets is the lowest synset in the WordNet hierarchy that both input synsets have as a hypernym. This measure uses the ratio between the depth of the LCS and the sum of the depth of both synsets respectively.

Jiang & Conrath [22] uses the concept of information content (IC). The information content of a synset is related to the probability that a random word is an instance of that synset. The approach by Jiang & Conrath uses the IC from both input synsets and the IC of their LCS to calculate similarity.

Extended Lesk [23] utilizes all relations in WordNet that are of lexical-semantic nature and the concept of gloss overlap. The gloss overlap of two synsets s_1, s_2 is the amount of words that occur in the textual definition of both synsets. To calculate similarity of synsets, this measure takes into account the gloss overlap of each synset that is adjacent to one of the input synsets.

Ezzikouri et al. [24] determines the similarity of synsets by counting how many words occur in both input synsets and how many words their glosses share.

3.1.3 Corpus-based similarity measures

This subsection presents WSMs that utilize information gained from analyzing corpora.

SEWordSim [25] is a word similarity database which was constructed by analyzing information from StackOverflow pages. The analysis utilizes the concept of word co-occurence and positive pointwise mutual information (PPMI) to convert words into vectors. Cosine similarity is then used to calculate similarity between vectors. Since all analyzed words come from StackOverflow posts, the inferred meanings and similarities of words are most useful when utilized in the domain of software engineering and less useful anywhere else.

word2vec [26] is the most popular way to construct vectors from words. A neural network analyses surrounding words (local context) for each occurence of a word in a corpus. There are two different training processes defined for this neural network. One of them, called the continuous bag of words (CBOW) process, is training the neural network to guess the most likely context of an input word. The other training process, called skip-gram, trains the network by having it guess the word that best fits a given input context.

fastText [27] is an extension of word2vec. Unlike word2vec, each word is represented as an unordered collection of n-grams. An n-gram of a word is just a fragment of that word with n characters. For example, both "bi" and "cy" are bigrams of the word "bicycle". The vector representation of a word can then be calculated by combining the vector representations of all n-grams that make up that word. This allows turning words into vectors that were previously unknown as long as these words are concatenations of previously known n-grams.

GloVe [28] is another popular vector construction algorithm. Whereas word2vec trains its model solely by utilizing the local context of words, GloVe utilizes both local context and global corpus statistics. It trains its model to minimize a certain loss function. Minimizing this loss function results in the dot product between two word vectors w and $\tilde{w}$ being roughly equal to the logarithm of the co-occurence count between the words.

3.1.4 Hybrid measures

This subsection covers measures that utilize information from both corpora and semantic networks.

Nasari [29, 30] is a hybrid approach that utilizes both corpus-based word embeddings and lexical networks for similarity computation. For each BabelNet synset b, relevant Wikipedia pages are collected. The contents of these pages are then combined into a single unordered collection of words $W(b)$. The authors describe three different ways to construct vectors from these word collections. One of them being the embed vector representation technique which requires an existing vector embedding E. The embed vector of a given BabelNet synset b is the weighted sum $\sum_{w_i \in W(b)} f_i * E(w_i)$ where $E(w_i)$ is the embedded word vector of a given word w_i and f_i is that word's assigned weight.

DeConf [31] is another hybrid approach utilizing existing knowledge-based and corpus-based techniques. The idea behind DeConf is to deconflate a word into its different meanings. This tackles the problem of two occurences of the same word being deemed similar, even though they each refer to a different meaning of the word. Deconf works by utilizing an existing word embedding and an existing semantic network. For example, word2vec and WordNet could be used for this purpose. The semantic network is used to calculate a set of so-called sense biasing words for each word sense. The idea behind these kinds of sets is that they "can effectively pinpoint the semantics of individual synsets [31]." The word embedding allows representing these biasing words as numeric vectors. Each word sense can then be represented as a weighted sum of each biasing word vector. Compared to Nasari, which uses Wikipedia as its corpus-based source, Deconf can utilize any corpus. This allows for a greater number of unique words and also types of words that are usually not represented as pages on Wikipedia, like verbs and adjectives.

3.2 Related Traceability Link Recovery Work

Several literature reviews regarding TLR were performed covering works from 1999 to 2016 [5, 32, 33]. These reviews show that the most common TLR techniques are based on Information Retrieval (IR). IR is concerned with providing relevant documents given a search query. Documents in IR are often represented as an unordered collection of words [5]. The field of TLR can utilize IR techniques by treating artifacts as documents and queries.

Most TLR techniques are based on one of the following approaches: Vector Space Models, Latent Semantic Indexing, and probabilistic models. *Vector Space Models* (VSMs) transform documents and queries into numeric vectors. Each component of a vector represents a unique term that either occurs or does not occur in the document. The value of a vector component represents a weight that reflects the importance of that term to the document. To find the relevant documents for a given query, the vector representation of the query is compared to the vector representations of the documents. This comparison involves calculating how many terms occur in both the query and the document while also taking into account specific weights for each term. The idea behind this approach is that the most relevant documents for a query will have vector representations that are the most similar to the query vector. *Latent Semantic Indexing* (LSA) [34] is an extension of VSM. With this technique, documents and queries are also transformed into vectors. The difference here is, that the dimension of the used vector space is reduced and each individual dimension represents something that can be interpreted as a concept rather than a specific term. Consequently, similar terms are grouped into the same concept and thus the same dimension. Therefore, instead of dealing with individual terms when comparing documents with the query, this technique calculates how many concepts occur in both the query and the document. *Probabilistic Models* work by calculating the probability of how relevant a certain document is to a given query. To do this, they utilize probabilistic models that allow calculating $\mathbb{P}(D_i \mid Q)$ which is the probability that the document D_i is related to the query Q.

For further improvements in performance, TLR approaches often use so-called enhancement strategies in addition to the approaches mentioned above. Borg et al. [5] mention the most popular enhancement strategies. One strategy is the use of a thesaurus to assist in the synonymy problem. A thesaurus is a resource that acts like a word similarity measure by providing synonyms and related terms for a given word. Settimi et al. [35] assess the usage of a thesaurus in their TLR technique. Their evaluation showed overall slightly worse results using a thesaurus which, according to the authors, might be caused "by the thesaurus not being able to capture context-specific concepts" [35]. Hayes et al. [36] and Leuser et al. [37] also assess the effects of utilizing a thesaurus and report better results with the usage of a thesaurus. Guo et al. [38] uses deep learning techniques and the skip-gram word embedding for TLR between requirements and design artifacts. The skip-gram word embedding is trained over a large set of domain specific documents to specifically learn the meanings of words in the context of the relevant domain.

While these works all cover the effects of word similarity measures on TLR, they either don't explicitly mention which measures or resources are used, or they only cover one measure or resource each. In contrast, this thesis aims to cover a wide range of WSMs and focuses on the effects of the WSMs on TLR performance.

4 Implementation

This chapter explains how and which word similarity measures were implemented in ArDoCo. The following section explains the decision process behind choosing WSMs from literature and implementing them in ArDoCo. Section 4.2 presents the part of ArDoCo's architecture that allows the inclusion of multiple WSMs in the TLR process.

4.1 Choosing Word Similarity Metrics

The following criteria were taken into account when choosing which word similarity measures should be implemented and evaluated.

- *Variability*: Different categories of WSMs have different strengths and weaknesses. For example, while lexical measures can identify typos very well, they are unable to identify semantic relatedness between different looking words as well as corpus-based or knowledge-based WSMs can. A wide variety of WSMs should be chosen to benefit from the strengths of all types of WSMs while minimizing the weaknesses.
- *Popularity*: WSMs that are well known in the literature probably perform better than unpopular WSMs since their effectiveness is likely the reason they are well known to begin with. Popular WSMs also tend to have better availability when it comes to useful artifacts like pre-trained data sets or implementations in programming languages. Therefore, WSMs that are often mentioned in literature should be preferred.
- *Modernity*: Newer WSMs have various potential advantages over older measures. They are able to rectify flaws of their precedessors, make use of increased processing power and take advantage of more data and knowledge in the field of natural language processing.

Using these criteria, the following word similarity measures were chosen to be implemented and evaluated:

- *Levenshtein distance* [11]: This measure was already part of ArDoCo's TLR process prior to this thesis's work. No additional implementation work was required.
- *Jaro Winkler* [14]: This measure was also already part of ArDoCo's TLR process.
- *N-Gram by Kondrak* [10]: Kondrak's measure establishes a family of lexical WSMs with one member of this family being the Levenshtein distance. Since Kondrak's evaluation indicates other members of this family to be superior, it could potentially replace Levenshtein and improve the TLR process in ArDoCo.

- *SEWordSim* [25]: SEWordSimDB is a corpus-based WSM that was constructed by analyzing data from StackOverflow posts. This allows the WSM to find word pairs that can only be identified as similar in the context of the software engineering domain. While other measures were trained on general purpose text, this focus on the domain of software engineering might provide a unique strength.
- *fastText* [27]: Since fastText is a corpus-based measure that is trained on n-grams, it could potentially provide vector embeddings and thus similarity scores for word pairs that were not in the training corpus.
- *GloVe* [28]: Another corpus-based measure that, in contrast to the local context-based training of fastText, uses global co-occurence information to compute word similarity.
- *NASARI* [29, 30]: The only hybrid measure evaluated in this thesis. NASARI is both knowledge-based and corpus-based and gives an idea of how well hybrid measures can perform.
- *WordNet* [39]: Being the only semantic network in this thesis, WordNet gives an idea of how lexical networks can aid the TLR process. The following WordNet algorithms were chosen:
  - *Leacock & Chodorow* [20]: Bases its similarity score on the distance between two synsets on the WordNet graph.
  - *Jiang & Conrath* [22]: Uses the concept of information content of two synsets to compute their similarity.
  - *Extended Lesk* [23]: Calculates the gloss overlap between the neighbourhood of two synsets.
  - *Ezzikouri* [24]: A more modern measure that, in addition to utilizing gloss overlap, also uses the words in the synsets themselves to calculate similarity. Since the authors have not performed an evaluation for this measure, it would be interesting to see how it compares to the other WordNet algorithms.

  All of these measures are different in that they utilize different kinds of information that is present in the WordNet graph.

4.2 Architecture

The complete architecture of ArDoCo's TLR process is described in further detail in Keim et al. [2] and the current state of it can be viewed on the public GitHub repository [8]. The part that is relevant for this thesis mainly consists of the classes seen in Figure 4.1.

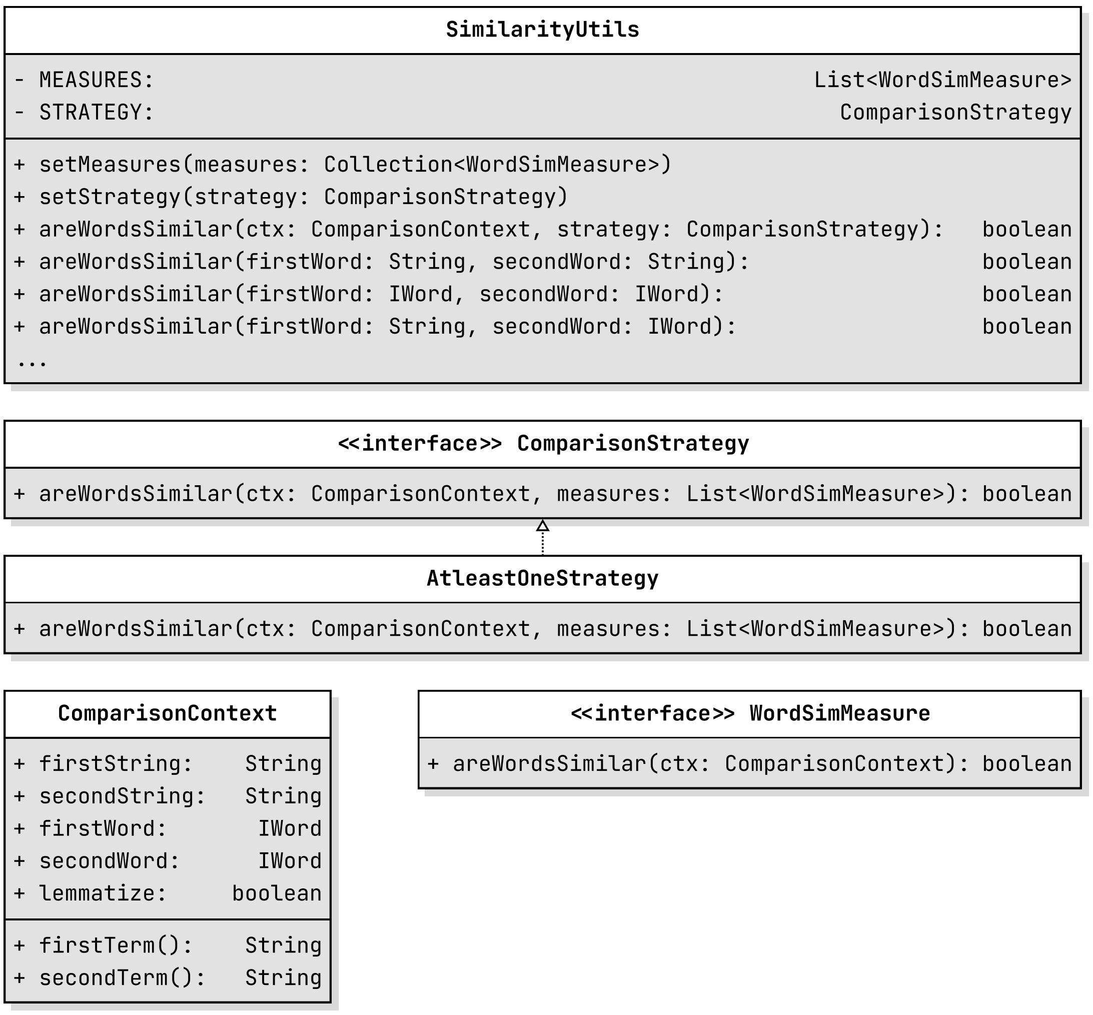

SimilarityUtils
- MEASURES: List<WordSimMeasure>
- STRATEGY: ComparisonStrategy
+ setMeasures(measures: Collection<WordSimMeasure>)
+ setStrategy(strategy: ComparisonStrategy)
+ areWordsSimilar(ctx: ComparisonContext, strategy: ComparisonStrategy): boolean
+ areWordsSimilar(firstWord: String, secondWord: String): boolean
+ areWordsSimilar(firstWord: IWord, secondWord: IWord): boolean
+ areWordsSimilar(firstWord: String, secondWord: IWord): boolean
...
<<interface>> ComparisonStrategy
+ areWordsSimilar(ctx: ComparisonContext, measures: List<WordSimMeasure>): boolean
AtleastOneStrategy
+ areWordsSimilar(ctx: ComparisonContext, measures: List<WordSimMeasure>): boolean
ComparisonContext
+ firstString: String
+ secondString: String
+ firstWord: IWord
+ secondWord: IWord
+ lemmatize: boolean
+ firstTerm(): String
+ secondTerm(): String
<<interface>> WordSimMeasure
+ areWordsSimilar(ctx: ComparisonContext): boolean

Figure 4.1: The main classes of ArDoCo's WSM architecture

A `ComparisonContext` is meant to contain all necessary information to recognize whether two words are similar. Instances of the `IWord` class provide additional information like the corresponding sentence, a word's lemma or its part of speech tag. Since not all comparisons involve `IWord` instances, the fields `firstWord` and `secondWord` are nullable. WSMs can utilize this additional information in cases where one of these fields is set. A `ComparisonStrategy` determines how the verdicts of multiple WSMs regarding a specific comparison are combined. The `AtleastOneStrategy` works by treating a word pair as similar when at least one of the used WSMs considers the word pair as similar. The `SimilarityUtils` class is the main access point for word similarity comparison. It provides multiple static methods that allow easy comparison between strings or `IWord` instances. This class allows customizing how it performs its comparisons, by either passing a custom strategy to one of its methods, or by setting the privately stored strategy and measures with the `setMeasures()` and `setStrategy()` methods. Calls to the `SimilarityUtils` class happen throughout various points in ArDoCo's TLR process. Any calls for word comparisons without a specified strategy, will default to the already stored strategy. For each call, the class passes its stored WSMs to the appropriate `ComparisonStrategy`.

This architecture allows dynamically changing which WSMs are involved and how they are combined. Each implemented WSM can additionally be configured through a configuration file. This configuration file provides individual settings for each WSM and allows enabling or disabling any WSM for the TLR process.

5 Evaluation

This chapter covers the evaluation performed on the various WSMs that were implemented in ArDoCo. The goal of this evaluation is to answer the research questions stated in the introduction:

- How do word similarity measures affect the performance of TLR between natural language SAD and formal architectural models?
- Which kinds of WSMs are particulary effective or ineffective when used for TLR?
- What kinds of problems are encountered when utilizing WSMs for TLR?

The first section of this chapter explains how this evaluation was approached. It establishes the metrics used to quantify the effects of WSMs that are used to answer the first and second research questions. The second and third sections present the case studies and baselines that were used. Section 5.4 presents the results of each evaluated WSM along with any encountered problems. The final section provides an overview of the results by comparing the evaluated WSMs against each other. This overview provides an answer to the second research question. Additionally, the final section discusses various aspects of this evaluation, such as additional metrics, reliability, validity, and scope.

5.1 Methodology

To assess the impact of word similarity measures on TLR performance, several case studies are utilized. Each case study contains a model, textual SAD and a gold standard with all correct trace links between these two artifacts. The gold standards allow us to classify found trace links as true positives or false positives, depending on whether the found link exists in the gold standard. Missing trace links are classified false negatives if they exist in the gold standard, or as true negatives if they don't. Using this classification scheme, the metrics *precision*, *recall*, and F_1*-Score* can be calculated. A single evaluation for a specific WSM consists of running ArDoCo's TLR process on each case study, only having the relevant WSMs enabled. Afterwards, the sums of all true positives, false positives, true negatives, and false negatives from all case studies are calculated to determine the F_1-Score, precision, and recall for the evaluated WSM. This is effectively a weighted average of the F_1, precision, and recall scores from all case studies where case studies with more sentences and model elements have a bigger impact on the final score. As for the comparison strategy, the `AtLeastOne` strategy was used for all evaluations (cf. Section 4.2).

5.2 Case studies

There are four existing case studies for ArDoCo:

- *Mediastore* (MS): A web-based software system that acts as a store where music can be uploaded, downloaded and bought [40].
- *Teammates* (TM): A cloud-based tool for sharing feedback between students and teachers [41].
- *TeaStore* (TS): A micro-service web store for tea and tea supplies [42].
- *BigBlueButton* (BBB): A web conferencing system for online learning [43].

These case studies can be found on the public GitHub repository for ArDoCo [8]. Each of them have different numbers of words, sentences, and model instances. Table 5.1 shows some of the differences between the case studies.

	$\sum$	BBB	MS	TM	TS
#Model Instances	45	12	14	8	11
#Sentences	363	85	37	198	43
#Words	2030	705	330	1107	416
#Trace Links	192	50	29	84	29

Table 5.1: General statistics about the used case studies

5.3 Baselines

There are two baselines for this evaluation.

The first baseline (BASE1) represents the TLR process in ArDoCo where only equality comparisons between words are performed. No word similarity measure is used for this baseline. Since this thesis aims to assess the impact a WSM has on the TLR process, no other WSM should interfere with this impact analysis during evaluation. Evaluating the first baseline yields the results shown in Table 5.2.

Metric	Overall	BBB	MS	TM	TS
Precision	91.39%	87.87%	100.00%	89.02%	100.00%
Recall	72.63%	58.00%	58.62%	87.95%	67.85%
F_1	80.93%	69.87%	73.91%	88.48%	80.85%

Table 5.2: Evaluation results for the first baseline

The second baseline (BASE2) represents the TLR process in ArDoCo with Levenshtein distance and Jaro Winkler utilized for similarity comparisons. These two measures were already implemented into ArDoCo before the work for this thesis was performed. The information gained with this baseline is of interest for the continued improvement of ArDoCo itself. Both measures were used with their default configuration values: threshold=0.9, minLength=2, maxDistance=1. These configuration values are explained in Subsection 5.4.1 and Subsection 5.4.2. The evaluation for the second baseline yields the results shown in Table 5.3.

Metric	Overall	BBB	MS	TM	TS
Precision	90.53%	85.71%	100.0%	89.02%	100.0%
Recall	80.52%	84.00%	62.06%	87.95%	71.42%
F_1	85.23%	84.84%	76.59%	88.48%	83.33%

Table 5.3: Evaluation results for the second baseline

The upcoming evaluation results are mostly concerned with the first baseline but will also mention how the implemented measures fare against Levenshtein distance and the Jaro Winkler measure. All F_1, precision, and recall results shown are based on one of the two baselines. All results based one the first baseline were generated by running ArDoCo's TLR process with only the evaluated WSM enabled. All results based on the second baseline were generated with the evaluated WSM, the Levenshtein measure, and the Jaro Winkler measure enabled.

5.4 Results

The following subsections present the evaluation results for each implemented word similarity measure. Some of these WSMs are configurable such that different configuration values yield different evaluation results. For these circumstances, the relevant configuration values are mentioned. The results will be presented mostly as graphs but the exact values are also available on Zenodo [44].

5.4.1 Jaro Winkler

Since Jaro Winkler is part of the second baseline, the evaluation for this measure only consists of a comparison to the first baseline. The only configurable value for this measure is the similarity threshold. Whenever the WSM calculates a similarity score for a word pair, it compares that score with the similarity threshold. If the similarity score is greater than the threshold, the WSM will recognize the word pair as similar. This similarity threshold value also exists for most of the subsequent WSMs and affects precision and recall.

Figure 5.1 shows the F_1-Score, precision, and recall values for each similarity threshold that it was evaluated on. The dashed lines represent the F_1-Score, precision, and recall of the first baseline.

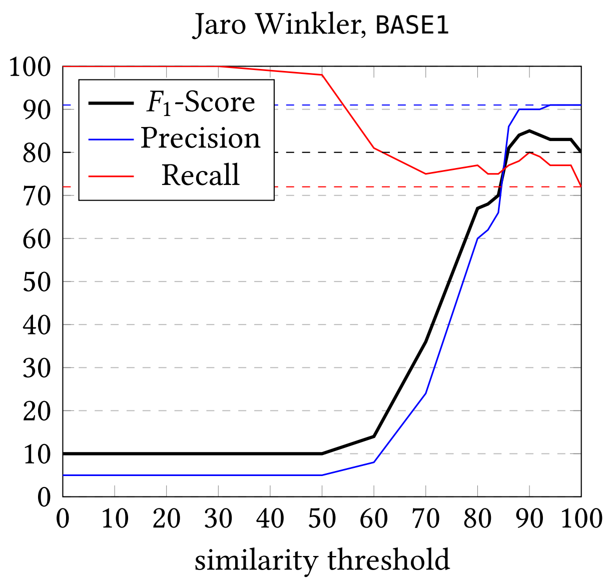


Figure 5.1: Jaro Winkler evaluation graph

Figure 5.1 shows that this WSM is particulary effective with similarity thresholds above 80% with the F_1-Score reaching its maximum point at a threshold of about 90%. Common false negatives are coreferences, different looking synonyms, and abbreviations like "DB".

5.4.2 Levenshtein

Since the Levenshtein measure is also part of the second baseline, the evaluation for this measure will also only consist of a comparison to the first baseline. This measure has three configurable values:

1. *Max Distance*: Word pairs with a Levenshtein distance above this configuration value will not be considered similar.
2. *Min Length*: If one of the words is shorter than this configured value, an additional condition must be met for the word pair to be considered similar. This condition being that one word must contain the other.
3. *Threshold*: A number between zero and one that serves as a word-dependent distance limit. The levenshtein distance between the words must be lower than the threshold multiplied by the length of the shorter word.

With a max distance and min length both ranging from 0 to 12 and a threshold ranging from 0.0 to 1.0 with increments of 0.1, all 1859 possible combinations of these three configuration values were evaluated. The resulting precision and recall pairs can be seen in Figure 5.2. The dashed lines in this figure represent the precision and recall of the first baseline. This figure shows that for precision values above 80%, the recall has an upper limit of 78%. The overall best result with a precision of 91% and a recall of 78% is colored in red. The results colored in yellow represent the pareto set. Fifteen configurations produced this result: All configurations with a threshold of 20%, zero min length, and a max distance greater than one, and all configurations with a threshold of 20%, a min length less than five, and a distance of two.

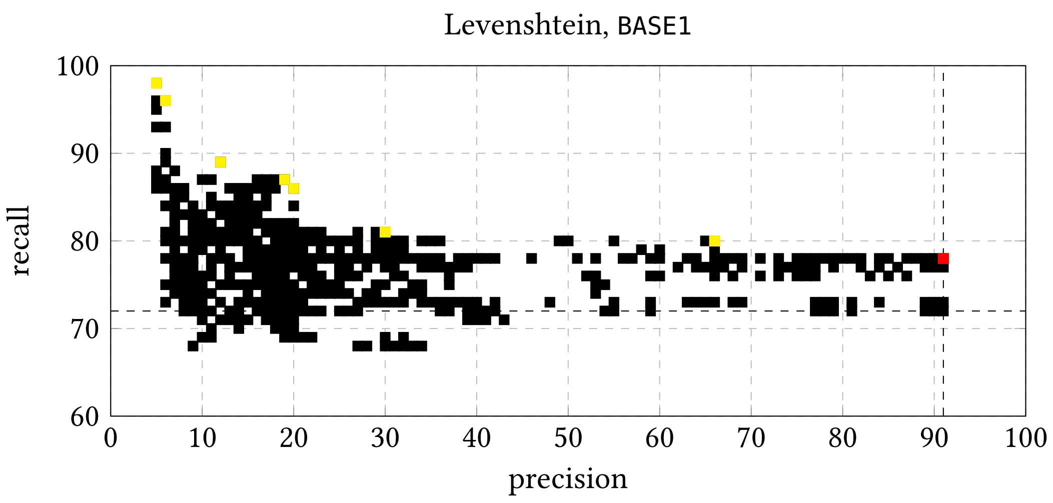


Figure 5.2: The precision and recall pairs of all evaluated Levenshtein configurations

5.4.3 N-gram

Kondrak's N-gram measure was evaluated on different values of n and with different similarity thresholds. Kondrak's evaluation suggests that there is no significant difference between the distance and similarity variant [10]. Therefore, the distance variant was used for this evaluation.

Figure 5.3 provides a comparison of Kondrak's measure with values of n ranging from two to six. It shows that while precision seems to increase with greater lengths of n, recall reaches a plateau at a threshold of about 60%. Although no value of n performs significantly better than any other, the precision of the $n = 2$ variant seems to consistently trail behind the other variants. Optimal precision and recall seem to be at thresholds around 80%. Figure 5.4 shows a more detailed evaluation of the $n = 3$ case regarding both baselines.

Comparing it with the second baseline, the inclusion of the N-gram measure seems to provide little to no improvement. This is most likely because Levenshtein distance and Jaro Winkler cover most of what lexical WSMs are able to do. As expected with lexical measures, Ngram is also unable to recover trace links where coreferences, different looking synonyms or abbreviations are used. Word pairs like "DB" and "database" are missed.

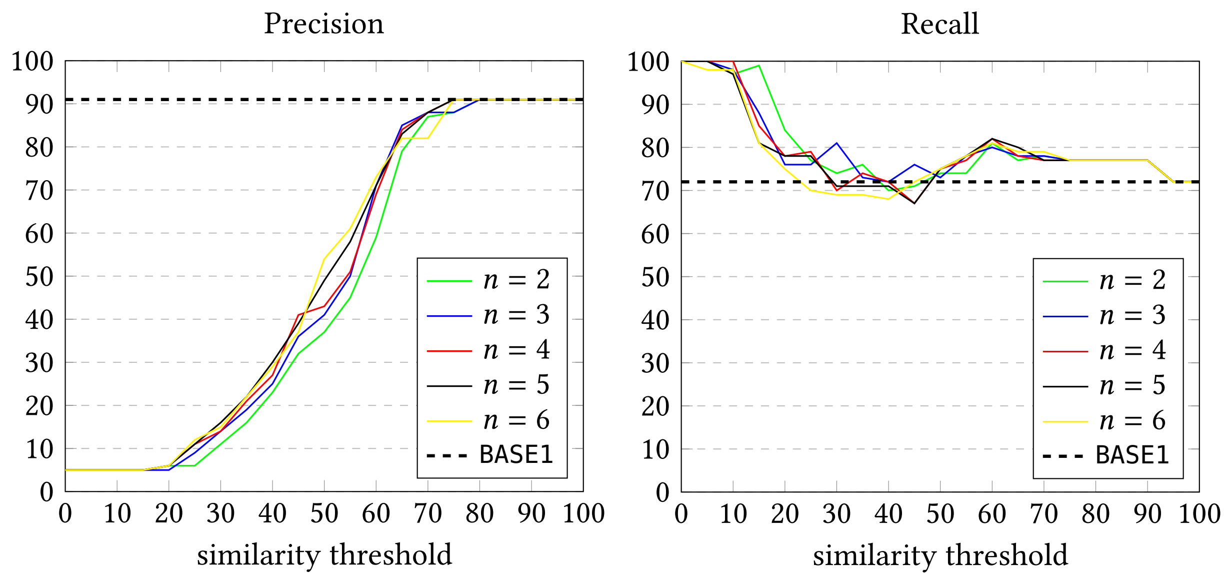


Figure 5.3: Comparing different values of n for Kondrak's N-gram WSM

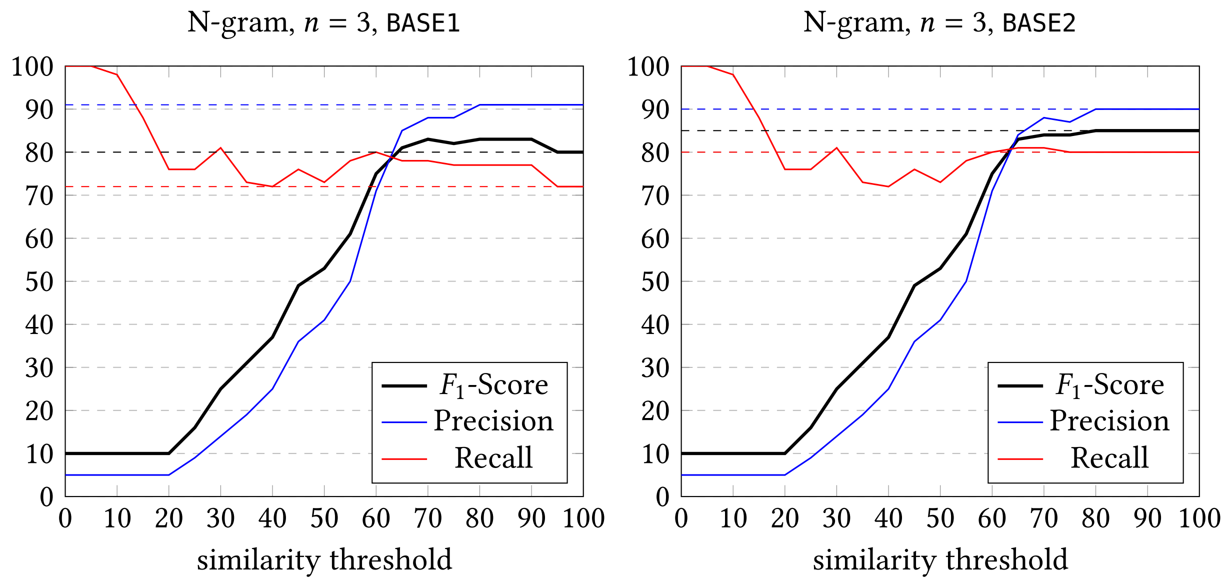


Figure 5.4: N-gram evaluation graphs for $n = 3$

5.4.4 SEWordSim

For the evaluation of the SEWordSim measure, the same sqlite database from Tian et al. [25] with 5 636 534 pairs of words was used. Each database entry is a pair of words combined with a pre-computed similarity score. As with some of the previous measures, the only configurable value for this WSM is the similarity threshold. Figure 5.5 shows the evaluation graph for this measure regarding both baselines.

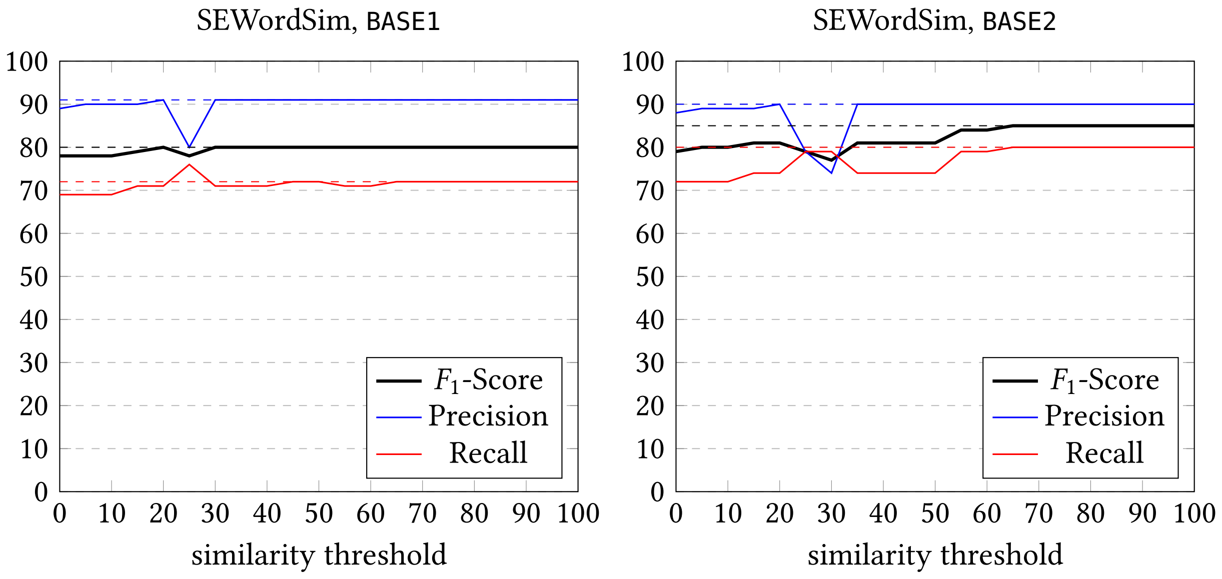


Figure 5.5: SEWordSim evaluation graphs

Figure 5.5 shows no significant performance improvements compared to either baseline. This is most likely due to the large number of word pairs with no pre-calculated similarity score. During a single evaluation, around 92 394 unique word pairs are looked up in the database. Of those 92,394 word pairs, only 5428 are found. This results in SEWordSim only being able to have an impact on TLR in five percent of all comparisons.

5.4.5 fastText

The fastText measure was evaluated on three binary models from the official fastText website [45]:

- wiki-news-300d-1M-subword (`WIKI`): One million word vectors trained on Wikipedia, UMBC webbase corpus and statmt.org news dataset.
- crawl-300d-2M-subword (`CRAWL`): Two million word vectors trained on Common Crawl.
- cc.en.300 (`CC`): Word vectors trained on Wikipedia and Common Crawl.

All of these pre-trained models consist of 300 dimensonal vector embeddings for n-grams of various lengths. Figure 5.6 provides a comparison of these models.

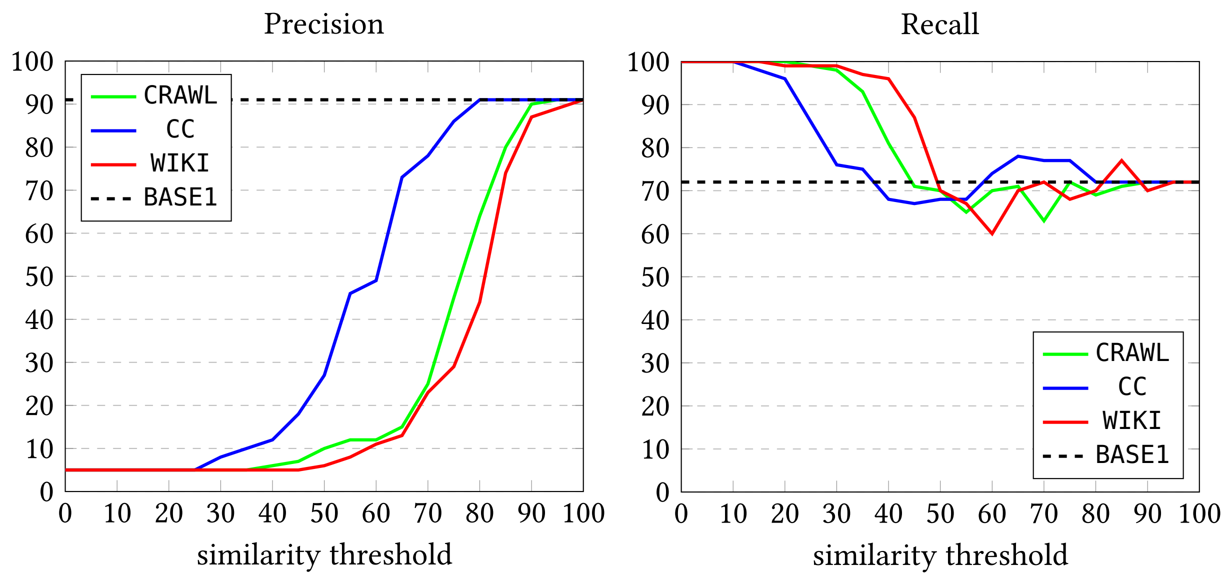


Figure 5.6: Comparison of different fastText models

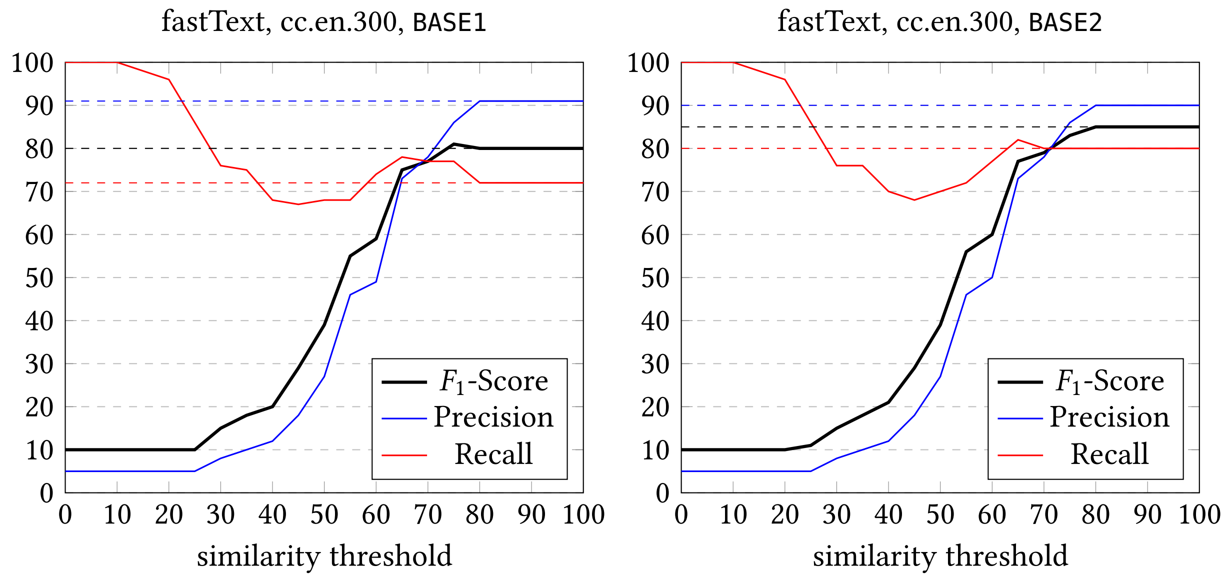


Figure 5.7: fastText cc.en.300 evaluation graphs

Figure 5.6 shows that the cc.en.300 (`CC`) model seems to be the only model providing improved recall at precision scores above 80%. Figure 5.7 presents a more detailed evaluation of the cc.en.300 model. The highest recall value while maintaning a high precision value seems to be achieved at a threshold around 65%. No significant improvement regarding the second baseline can be seen.

5.4.6 WordNet

Four different WordNet based word similarity algorithms were used for this evaluation. Each of them have a different approach for comparing concept similarity on the WordNet graph. Figure 5.8 provides a comparison of these algorithms.

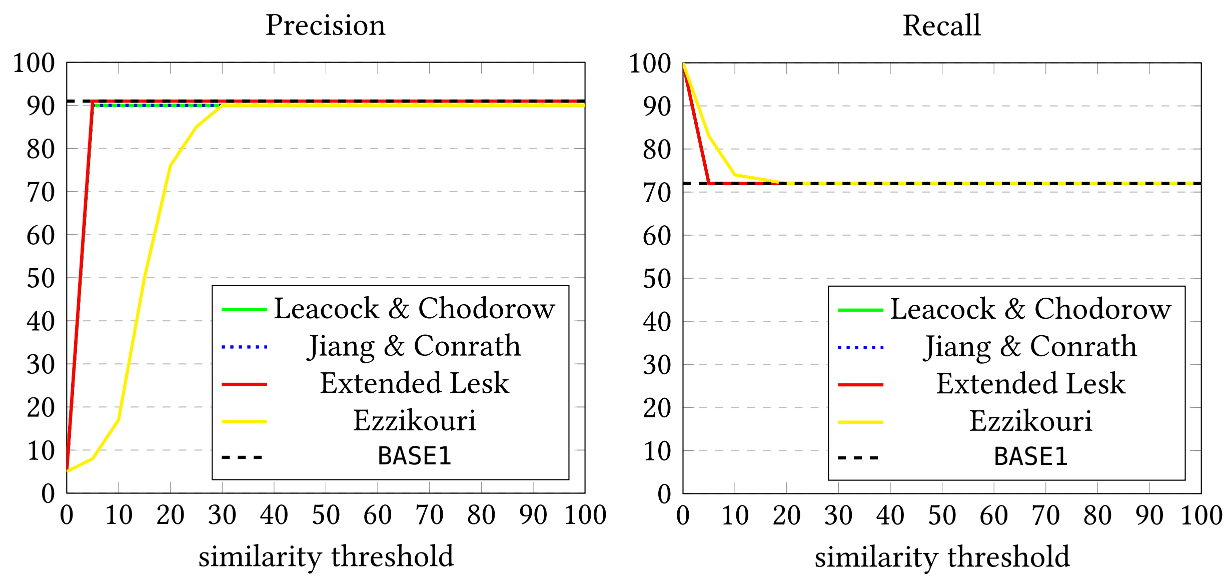


Figure 5.8: Comparison of WordNet algorithms

As Figure 5.8 shows, none of the algorithms have a positive effect on TLR performance. At thresholds above 10%, the algorithms seem to be able to identify word pairs like “CPU” and “Processor” or pairs like “data” and “information” as similar. At least with the used gold standards, these kinds of words do not seem to be involved in architectural trace links. Regarding the second baseline, no improvement of recall or precision is seen either.

5.4.7 GloVe

GloVe was evaluated using the pre-trained vector embeddings provided on the GloVe website: Common Crawl (`CC`) with 2.2 million words, Wikipedia + Gigaword (`WIGI`) with 400 thousand words and Twitter (`TWTR`) with 1.2 million words. While the `CC` embeddings are vectors with 300 dimensions, the other two embeddings are available with multiple different dimensions. Figure 5.9 provides a comparison of the best performing embeddings.

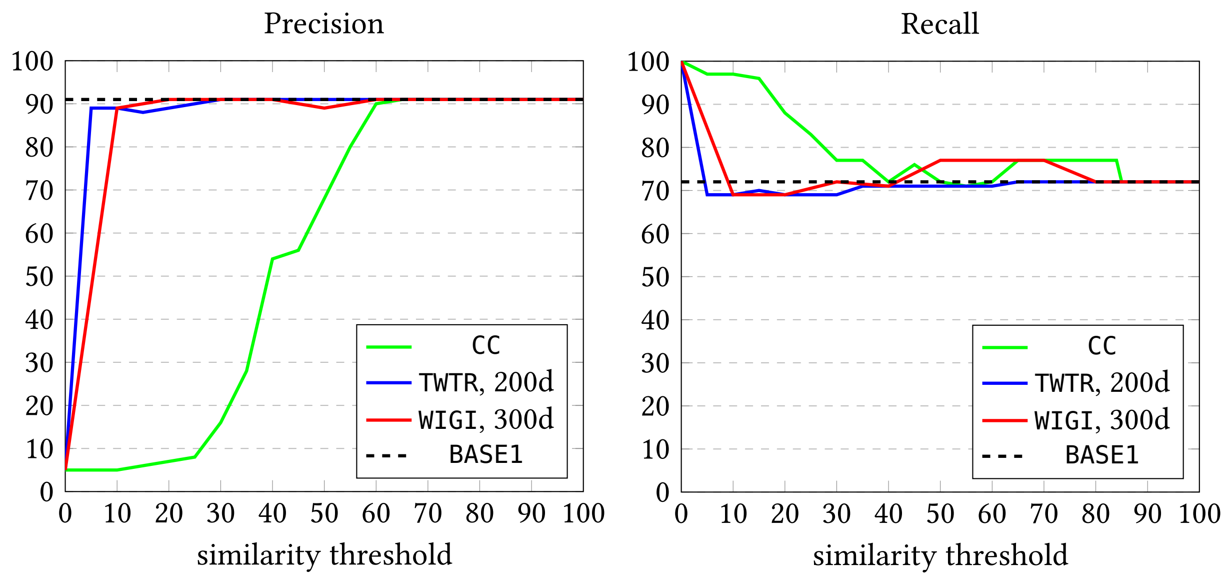


Figure 5.9: Comparison of the GloVe vector embeddings

Figure 5.9 shows that the Wikipedia + Gigaword vector embeddings performed the best although the Common Crawl embeddings perform about the same at higher similarity thresholds. Evaluating variants of the embeddings with lower dimensions yielded slightly worse results. Figure 5.10 shows the evaluation results of the `WIGI` embedding regarding both baselines.

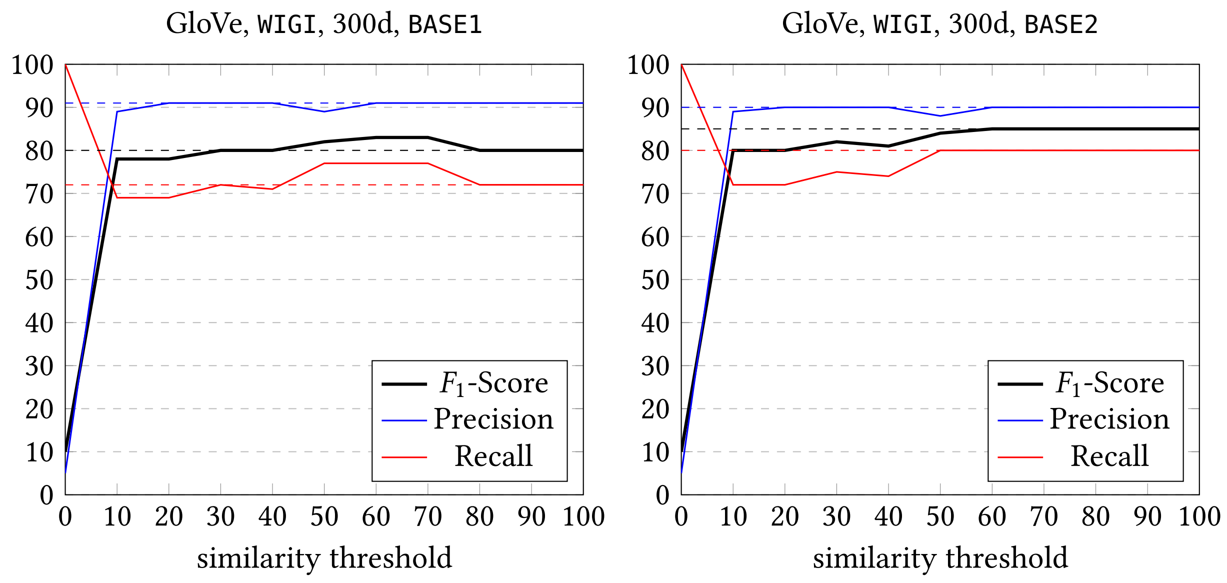


Figure 5.10: GloVe evaluation graphs using the 300 dimensional `WIGI` embeddings

5.4.8 NASARI

Nasari was evaluated using two pre-trained word2vec embeddings from the offical Nasari website [46]: the english embed vectors trained on the Google News dataset containing 100 billion words (`NEWS`), and word embeddings trained on the UMBC WebBase corpus containing three billion english words (`UMBC`). For each input word, the Nasari measure queries the BabelNet API for a collection of BabelNet synsets that contain the input word. Synsets from the first word are then compared with synsets from the second word by calculating the cosine similarity of their respective vector representations. A word pair is considered similar, if the cosine similarity between one of the compared synsets exceeds the similarity threshold. Figure 5.11 shows the evaluation results for both embeddings regarding both baselines. The dotted lines represent the results for the `UMBC` embedding.

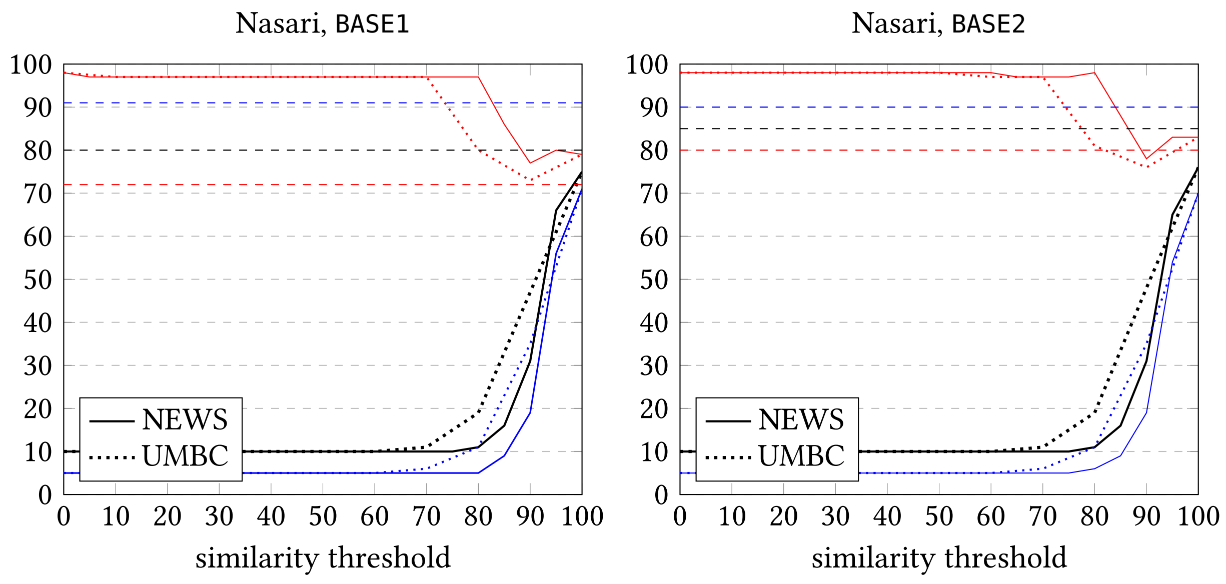


Figure 5.11: Nasari evaluation graph

Figure 5.11 shows that the Nasari measure never reaches the F_1 level of either baseline, regardless of which embedding is used. The Google News embedding seems to have slightly better precision while the UMBC embedding has slightly better recall. At thresholds below 90%, Nasari seems to regard many unrelated word pairs like “Logic” and “configuration” (52% similarity) or “Server” and “html5” (76% similarity) as similar. Nasari vectors in general seem to have a much higher similarity than vectors from aforementioned measures. A major problem for Nasari is that some very loosely related words receive a similar score of 100%. For example, the words “Driver” and “home” have a similarity score of 100%. This is because the BabelNet synset `bn:17639563n` represents the album “Home” by the american band “Blue October” whose second track is titled “Driver”. The same synset comes up when querying BabelNet for either of these two words, which is why the resulting similarity is 100%.

5.5 Discussing the results

This section discusses various aspects of this evaluation. The following subsection provides an overview of the results of all evaluated WSMs. Subsection 5.5.2 explains why the metric *accuracy* did not provide meaningful insight for this evaluation. The last three subsections discuss reliability, validity, and scope.

5.5.1 Overview

Figure 5.12 provides a comparison between all evaluated WSMs each using their optimal configuration values[1]. With the exception of WordNet, all evaluated WSMs have managed to improve the recall relative to the first baseline. These improvements in recall all come with different amounts of precision loss. Overall, the lexical measures (Jaro Winkler, Levenshtein, and Ngram) provide the most increase in recall with minimal loss of precision. The purely knowledge-based measure WordNet seems to have performed the worst while the at least partially knowledge-based measure Nasari performed a bit better. The corpus-based measures (SEWordSim, fastText, GloVe) have performed relatively average. GloVe provides a slight improvement in recall while introducing no decrease in precision.

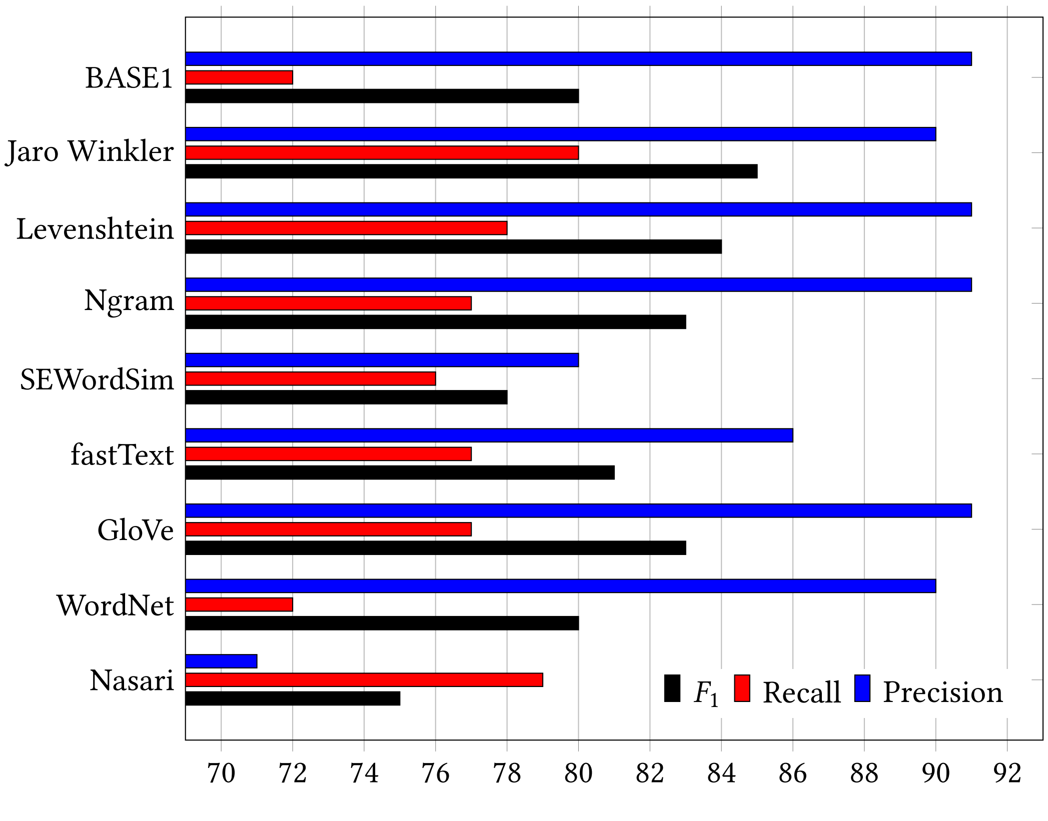


Figure 5.12: Comparison of all WSMs

[1]Jaro Winkler (threshold=90%), Levenshtein (threshold=20%, minLength=2, maxDistance=2), Ngram (threshold=75%, n=3), SEWordSim (threshold=25%), fastText (threshold=75%), WordNet (threshold=50%, any algorithm), GloVe (threshold = 60%, `WIGI` embedding), Nasari (threshold=100%)

5.5.2 Accuracy

Another recorded metric for this evaluation was accuracy. However, this metric does not provide any useful information about the TLR performance since the amount of false positives and true negatives heavily outnumber true positives and false negatives. This can be seen in Figure 5.13 where the rise in accuracy happens at the same thresholds where the false positives turn into true negatives. For this evaluation, accuracy nearing 100% does not imply good TLR performance since there are many more true negatives than true positives.

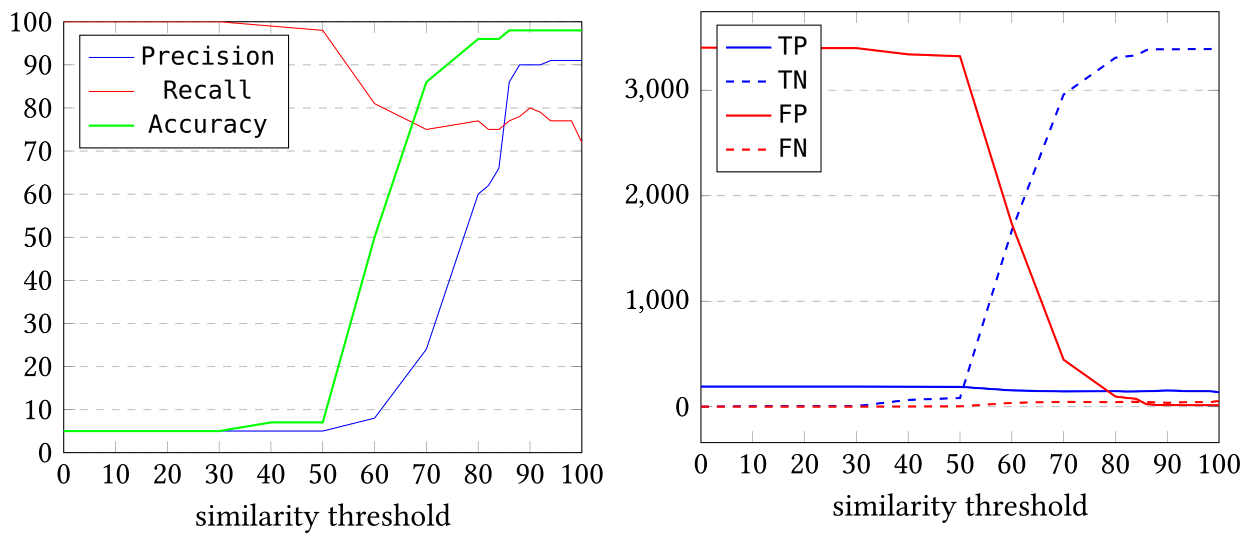


Figure 5.13: Extended Jaro Winkler Evaluation Graph

5.5.3 Reliability

The evaluation process for this thesis consists of executing ArDoCo's TLR algorithm for each case study and comparing the results to the previously mentioned baselines. The algorithm in itself is inherently deterministic and thus its results are reproducible. Therefore, the exact results should be reproducible if the exact same algorithm, used libraries, and data sources for the various WSMs are acquired. Herein lies the problem for this thesis's reliability.

The exact state of ArDoCo used for this evaluation can be acquired by checking out the "`thesis`" branch on the GitHub repository `github.com/kwerber/ArDoCo` [47]. Detailed steps to reproduce this evaluation can also be found in the repository.

Some of the investigated WSMs need seperate data sources to function. For example, for the evaluation of fastText, three different pre-trained binary models were used. These data sources were downloaded from various sites on the internet and may change or disappear. To ensure that such events do not impede the reproducibilty of this evaluation, all used data sources are also stored on Zenodo [44]. The detailed evaluation data and the state of the GitHub repository is also stored on Zenodo [44, 47].

5.5.4 Validity

The goal of this thesis was to assess the performance impact of word similarity measures on ArDoCo's TLR process. To accomplish this, various WSMs were selected, implemented, and their performance impact evaluated. Since the implementation of WSMs was the only change to ArDoCo during the evaluation process, all differences in precision and recall can only be attributed to the inclusion of the WSMs. Since nothing else could have affected the results, this thesis's evaluation should measure exactly what it was meant to measure.

5.5.5 Scope

There are five primary factors that limit the scope of this evaluation:

- Word similarity measures: The evaluation was performed only on a small number of WSMs. As shown in Section 2.2, modern WSMs can be classified as lexical, knowledge-based or corpus-based measures. For this evalation, three lexical WSMs, one knowledge-based, two corpus-based, and one hybrid approach was chosen.
- Case studies: The evaluation was performed on four different case studies. Two of them are example online stores that see no real use and, apart from TeamMates, all of the case studies have less than 100 sentences. Additionally, all of the case studies are web applications. While these case studies all describe software systems, they might not be representative for the whole range of possible software systems.
- ArDoCo: The only TLR-process evaluated was ArDoCo. Other approaches to TLR between natural language SAD and architectural models might yield different results.
- Language: All case studies and all pre-trained word models use the english language. TLR approaches and WSMs for different languages might yield different results.
- Comparison Strategy: All WSMs were evaluated using the `AtLeastOne` strategy. This is relevant for the second baseline, since the second baseline involves multiple WSMs at the same time. Since the first baseline only ever involves a single WSM at a time, the used strategy is irrelevant for the first baseline. However, other strategies where multiple WSMs are involved at the same time may produce better results.

6 Conclusion and Future Work

This thesis evaluated various word similarity metrics on ArDoCo's TLR process. The results of this evaluation show that WSMs provide a noticable but limited improvement to the TLR performance. Kondrak's NGram measure had the biggest increase in recall of about 10% and Nasari had the biggest decrease in precision of about 16%. Overall, the lexical measures have performed the best. The corpus-based measures performed slightly worse and the measures using semantic networks performed the worst. When it comes to the second baseline, whenever a WSM combined with the Levenshtein and Jaro Winkler measures improved recall, the precision would be considerable worse. Therefore, none of the second baseline evaluations have shown any improvements.

Different WSMs have encountered different problems. As expected, lexical WSMs are unable to identify different looking synonyms as similar. WordNet seems to mostly contain words that have had little effect on the recovery of trace links. Nasari uses BabelNet, which is a much bigger semantic network than WordNet. This led to the problem that unrelated words were considered synonymous, when BabelNet concepts exist that contain both words. This phenomenon can be attributed to the polysemy problem. If the measure would be able to more accurately guess the appropriate BabelNet concept based on the context of the word comparison, better results should be possible. The corpus-based measures seem to lie somewhere between the lexical and knowledge-based ones. While they are also affected by the polysemy problem, the magnitude of the effect is smaller compared to Nasari. Another encountered problem were coreferences. Of the 190 correct trace links that the gold standards provided, twelve of them could only be resolved through coreference resolution. However, this problem cannot be solved by word similarity measures. One possible way to improve the effect of WSMs is to explore different ways to combine multiple WSMs at the same time. The evaluation in this thesis only used the `AtLeastOne` strategy. Other more complex strategies where multiple specific WSMs have to agree under specific circumstances might yield better results.

Bibliography

[1] Jan Keim and Anne Koziolek. “Towards Consistency Checking Between Software Architecture and Informal Documentation”. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Hamburg, Germany: IEEE, Mar. 2019, pp. 250–253. ISBN: 978-1-72811-876-5. DOI: `10.1109/ICSA-C.2019.00052`. URL: `https://ieeexplore.ieee.org/document/8712160/` (visited on 12/11/2021).

[2] Jan Keim et al. “Trace Link Recovery for Software Architecture Documentation”. In: *Software Architecture*. Ed. by Stefan Biffl et al. Vol. 12857. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 101–116. ISBN: 978-3-030-86043-1 978-3-030-86044-8. DOI: `10.1007/978-3-030-86044-8_7`. URL: `https://link.springer.com/10.1007/978-3-030-86044-8_7` (visited on 11/30/2021).

[3] “ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary”. In: *ISO/IEC/IEEE 24765:2017(E)* (Aug. 2017). Conference Name: ISO/IEC/IEEE 24765:2017(E), pp. 1–541. DOI: `10.1109/IEEESTD.2017.8016712`.

[4] Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, eds. *Software and Systems Traceability*. London: Springer London, 2012. ISBN: 978-1-4471-2238-8 978-1-4471-2239-5. DOI: `10.1007/978-1-4471-2239-5`. URL: `http://link.springer.com/10.1007/978-1-4471-2239-5` (visited on 01/03/2022).

[5] Markus Borg, Per Runeson, and Anders Ardö. “Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability”. In: *Empirical Software Engineering* 19.6 (Dec. 1, 2014), pp. 1565–1616. ISSN: 1573-7616. DOI: `10.1007/s10664-013-9255-y`. URL: `https://doi.org/10.1007/s10664-013-9255-y` (visited on 11/30/2021).

[6] Roberto Navigli and Federico Martelli. “An overview of word and sense similarity”. In: *Natural Language Engineering* 25.6 (Nov. 2019), pp. 693–714. ISSN: 1351-3249, 1469-8110. DOI: `10.1017/S1351324919000305`. URL: `https://www.cambridge.org/core/product/identifier/S1351324919000305/type/journal_article` (visited on 11/30/2021).

[7] Nitesh Pradhan, Manasi Gyanchandani, and Rajesh Wadhvani. “A Review on Text Similarity Technique used in IR and its Application”. In: *International Journal of Computer Applications* 120 (June 18, 2015), pp. 29–34. DOI: `10.5120/21257-4109`.

[8] *ArDoCo Core GitHub*. URL: `https://github.com/ArDoCo/Core` (visited on 03/07/2022).

[9] Dhivya Chandrasekaran and Vijay Mago. “Evolution of Semantic Similarity - A Survey”. In: *ACM Computing Surveys* 54.2 (Feb. 18, 2021), 41:1–41:37. ISSN: 0360-0300. DOI: 10.1145/3440755. URL: https://doi.org/10.1145/3440755 (visited on 01/15/2022).

[10] Grzegorz Kondrak. “N-Gram Similarity and Distance”. In: *String Processing and Information Retrieval.* Ed. by Mariano Consens and Gonzalo Navarro. Red. by David Hutchison et al. Vol. 3772. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115–126. ISBN: 978-3-540-29740-6 978-3-540-32241-2. DOI: 10.1007/11575832_13. URL: http://link.springer.com/10.1007/11575832_13 (visited on 12/06/2021).

[11] Vladimir Iosifovich Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Soviet Physics Doklady* 10.8 (1965). Doklady Akademii Nauk SSSR, V163 No4 845-848 1965, pp. 707–710. URL: https://www.semanticscholar.org/paper/Binary-codes-capable-of-correcting-deletions%2C-and-Levenshtein/b2f8876482c97e804bb50a5e2433881ae31d0cdd (visited on 03/29/2022).

[12] Matthew A. Jaro. “Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida”. In: *Journal of the American Statistical Association* 84.406 (June 1, 1989), pp. 414–420. ISSN: 0162-1459. DOI: 10.1080/01621459.1989.10478785. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1989.10478785 (visited on 05/21/2022).

[13] Matthew A. Jaro. “Probabilistic linkage of large public health data files”. In: *Statistics in Medicine* 14.5 (1995), pp. 491–498. ISSN: 1097-0258. DOI: 10.1002/sim.4780140510. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.4780140510 (visited on 05/21/2022).

[14] William E. Winkler. *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.* 1990. URL: https://eric.ed.gov/?id=ED325505 (visited on 03/29/2022).

[15] George A. Miller and Walter G. Charles. “Contextual correlates of semantic similarity”. In: *Language and Cognitive Processes* 6.1 (Jan. 1, 1991). Publisher: Routledge, pp. 1–28. ISSN: 0169-0965. DOI: 10.1080/01690969108406936. URL: https://doi.org/10.1080/01690969108406936 (visited on 12/02/2021).

[16] Lingling Meng, Runqing Huang, and Junzhong Gu. “A Review of Semantic Similarity Measures in WordNet”. In: *International Journal of Hybrid Information Technology* 6.1 (2013), p. 12.

[17] A. Banu et al. “A Survey and Comparison of WordNet Based Semantic Similarity Measures”. In: *IJCST* Volume 4 (Issue 2 2013). URL: https://www.semanticscholar.org/paper/A-Survey-and-Comparison-of-WordNet-Based-Semantic-Banu-Fatima/c9b5b574beb0e522b0df4a1be85a9a5a1b171c36 (visited on 01/15/2022).

[18] Dr. K. Goyal. “Classification of Semantic Similarity Technique between Word Pairs using Word Net2019”. In: *International Journal of Engineering and Advanced Technology* 9 (Jan. 8, 2020), pp. 4397–4402. DOI: 10.35940/ijeat.B2961.129219.

[19] R. Rada et al. “Development and application of a metric on semantic nets”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 19.1 (Feb. 1989), pp. 17–30. ISSN: 00189472. DOI: 10.1109/21.24528. URL: http://ieeexplore.ieee.org/document/24528/ (visited on 12/18/2021).

[20] Claudia Leacock and Martin Chodorow. “Combining Local Context and WordNet Similarity for Word Sense Identification”. In: *WordNet: An Electronic Lexical Database*. Vol. 49. Jan. 1, 1998, p. 265. ISBN: 978-0-262-27255-1. DOI: https://doi.org/10.7551/mitpress/7287.003.0018.

[21] Zhibiao Wu and Martha Palmer. “Verb Semantics and Lexical Selection”. In: *arXiv:cmp-lg/9406033* (June 23, 1994). DOI: 10.3115/981732.981751. arXiv: cmp-lg/9406033. URL: http://arxiv.org/abs/cmp-lg/9406033 (visited on 12/18/2021).

[22] Jay J. Jiang and David W. Conrath. “Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy”. In: *arXiv:cmp-lg/9709008* (Sept. 20, 1997). arXiv: cmp-lg/9709008. URL: http://arxiv.org/abs/cmp-lg/9709008 (visited on 12/18/2021).

[23] Satanjeev Banerjee and Ted Pedersen. “Extended Gloss Overlaps as a Measure of Semantic Relatedness”. In: *IJCAI-2003* (May 13, 2003).

[24] Hanane Ezzikouri et al. “A New Approach for Calculating Semantic Similarity between Words Using WordNet and Set Theory”. In: *Procedia Computer Science*. The 10th International Conference on Ambient Systems, Networks and Technologies (ANT 2019) / The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019) / Affiliated Workshops 151 (Jan. 1, 2019), pp. 1261–1265. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.04.182. URL: https://www.sciencedirect.com/science/article/pii/S1877050919306490 (visited on 01/14/2022).

[25] Yuan Tian, David Lo, and Julia Lawall. “SEWordSim: software-specific word similarity database”. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ICSE ’14: 36th International Conference on Software Engineering. Hyderabad India: ACM, May 31, 2014, pp. 568–571. ISBN: 978-1-4503-2768-8. DOI: 10.1145/2591062.2591071. URL: https://dl.acm.org/doi/10.1145/2591062.2591071 (visited on 12/03/2021).

[26] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv:1301.3781 [cs]* (Sept. 6, 2013). DOI: 10.48550/arXiv.1301.3781. arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (visited on 12/17/2021).

[27] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (Dec. 2017), pp. 135–146. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051. URL: https://direct.mit.edu/tacl/article/43387 (visited on 01/08/2022).

[28] Jeffrey Pennington, Richard Socher, and Christopher Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (visited on 12/17/2021).

[29] José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. "NASARI: a Novel Approach to a Semantically-Aware Representation of Items". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Denver, Colorado: Association for Computational Linguistics, 2015, pp. 567–577. DOI: `10.3115/v1/N15-1059`. URL: `http://aclweb.org/anthology/N15-1059` (visited on 01/09/2022).

[30] José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. "Nasari: Integrating explicit knowledge and corpus statistics for a multilingual representation of concepts and entities". In: *Artificial Intelligence* 240 (Nov. 2016), pp. 36–64. ISSN: 00043702. DOI: `10.1016/j.artint.2016.07.005`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0004370216300820` (visited on 01/09/2022).

[31] Mohammad Taher Pilehvar and Nigel Collier. "De-Conflated Semantic Representations". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing.* Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. Austin, Texas: Association for Computational Linguistics, 2016, pp. 1680–1690. DOI: `10.18653/v1/D16-1174`. URL: `http://aclweb.org/anthology/D16-1174` (visited on 01/09/2022).

[32] Muhammad Atif Javed and Uwe Zdun. "A systematic literature review of traceability approaches between software architecture and source code". In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.* EASE '14. New York, NY, USA: Association for Computing Machinery, May 13, 2014, pp. 1–10. ISBN: 978-1-4503-2476-2. DOI: `10.1145/2601248.2601278`. URL: `https://doi.org/10.1145/2601248.2601278` (visited on 04/10/2022).

[33] Nasser Mustafa and Yvan Labiche. "The Need for Traceability in Heterogeneous Systems: A Systematic Literature Review". In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC).* Vol. 1. ISSN: 0730-3157. July 2017, pp. 305–310. DOI: `10.1109/COMPSAC.2017.237`.

[34] Scott Deerwester et al. "Indexing by latent semantic analysis". In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. ISSN: 1097-4571. DOI: `10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9` (visited on 01/04/2022).

[35] R. Settimi et al. "Supporting software evolution through dynamically retrieving traces to UML artifacts". In: *Proceedings. 7th International Workshop on Principles of Software Evolution, 2004.* Proceedings. 7th International Workshop on Principles of Software Evolution, 2004. ISSN: 1550-4077. Sept. 2004, pp. 49–54. DOI: `10.1109/IWPSE.2004.1334768`.

[36] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. "Advancing candidate link generation for requirements tracing: the study of methods". In: *IEEE Transactions on Software Engineering* 32.1 (Jan. 2006), pp. 4–19. ISSN: 1939-3520. DOI: `10.1109/TSE.2006.3`.

[37] Jörg Leuser and Daniel Ott. "Tackling Semi-automatic Trace Recovery for Large Specifications". In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Roel Wieringa and Anne Persson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 203–217. ISBN: 978-3-642-14192-8. DOI: 10.1007/978-3-642-14192-8_19.

[38] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. "Semantically Enhanced Software Traceability Using Deep Learning Techniques". In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). ISSN: 1558-1225. May 2017, pp. 3–14. DOI: 10.1109/ICSE.2017.9.

[39] George A. Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (Nov. 1, 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748 (visited on 12/16/2021).

[40] *Media Store - SDQ Wiki*. URL: https://sdqweb.ipd.kit.edu/wiki/Media_Store (visited on 05/11/2022).

[41] *TEAMMATES Developer Web Site*. original-date: 2014-05-02T07:43:00Z. May 10, 2022. URL: https://github.com/TEAMMATES/teammates (visited on 05/11/2022).

[42] *TeaStore*. original-date: 2017-08-18T06:22:29Z. Apr. 22, 2022. URL: https://github.com/DescartesResearch/TeaStore (visited on 05/11/2022).

[43] *BigBlueButton*. original-date: 2010-05-25T01:42:41Z. May 11, 2022. URL: https://github.com/bigbluebutton/bigbluebutton (visited on 05/11/2022).

[44] *Thesis Evaluation Data*. Zenodo. DOI: 10.5281/zenodo.6580280. URL: https://doi.org/10.5281/zenodo.6580280 (visited on 05/26/2022).

[45] *English word vectors · fastText*. URL: https://fasttext.cc/index.html (visited on 05/18/2022).

[46] *NASARI: a Novel Approach to a Semantically-Aware Representation of Items*. URL: http://lcl.uniroma1.it/nasari/#two (visited on 01/09/2022).

[47] *Thesis Evaluation Repository*. GitHub. DOI: 10.5281/zenodo.6583858. URL: https://github.com/kwerber/ArDoCo/tree/thesis (visited on 05/26/2022).