

An Information Theoretic Approach for Extracting and Tracing Non-functional Requirements

Anas Mahmoud
Computer Science and Engineering
Louisiana State University
Baton Rouge, LA 70803
mahmoud@csc.lsu.edu

Abstract—Non-functional requirements (NFRs) are high-level quality constraints that a software system should exhibit. Detecting such constraints early in the process is critical for the stability of software architectural design. However, due to their pervasive nature, and the lack of robust modeling and documentation techniques, NFRs are often overlooked during the requirements elicitation phase. Realizing such constraints at later stages of the development process often leads to architecture erosion and poor traceability. Motivated by these observations, we propose an unsupervised, computationally efficient, and scalable approach for extracting and tracing NFRs in software systems. Based on main assumptions of the cluster hypothesis and information theory, the proposed approach exploits the semantic knowledge embedded in the textual content of requirements specifications to discover, classify, and trace high-level software quality constraints imposed by the system's functional features. Three experimental systems are used to conduct the experimental analysis in this paper. Results show that the proposed approach can discover software NFRs with an average accuracy of 73%, enabling these NFRs to be traced to their implementations with accuracy levels adequate for practical applications.

I. INTRODUCTION

Non-functional requirements (NFRs) describe a set of quality attributes that a certain system should exhibit [1]. Such attributes enforce operational constraints on different aspects of the system's behavior such as its usability, security, reliability, performance, and portability [2]. Explicitly identifying NFRs early in the software process is critical for making initial design decisions, and later for evaluating architectural alternatives for the system [3]. However, NFRs are often overlooked during the system's requirements elicitation phase, where the main emphasis is on getting the system's functional features explicitly and formally defined [4]. Part of this phenomenon can be attributed to the vague understanding of what NFRs actually are, and the lack of effective NFRs elicitation, modeling, and documentation methods [5], [6], [7].

Failure to identify NFRs early in the process impacts the overall quality of the system [5], [8]. In particular, NFRs tend to be connected through various interdependencies and trade-offs that span over multiple modules in the solution space [9]. Therefore, if not accounted for in the initial design phase, integrating such quality constraints into the system's architecture at later stages of the development process can lead to architecture anomalies and erosion problems [3], [10].

Furthermore, the lack of explicitly defined NFRs complicates the process of mapping, or tracing, such high-level quality constraints to their low-level implementations [11], [12]. This complicates basic software testing and maintenance activities such as system verification and validation and change impact analysis [7].

Motivated by these observations, we propose a novel unsupervised approach for extracting and tracing non-functional software requirements. The main assumption is that, even if they are not explicitly defined, NFRs tend to be embedded in the functional requirements of the system [12]. For instance, a system with a *login* feature is assumed to be secure. However, such information is typically scattered over the system's requirements specifications. The proposed approach exploits the basic assumptions of the cluster hypothesis and information theory to capture and classify such information into various types of NFRs. In particular, individual keywords are initially extracted from the system's requirements specifications. The pairwise semantic distance between these words is then calculated based on their co-occurrence patterns in massive repositories of textual human knowledge [13]. The main assumption is that words that occur together tend to be related [14]. Clustering is then used to group these words into semantically-coherent clusters. The generated clusters are classified into different categories of NFRs based on their semantic similarity to basic NFRs labels.

In the second phase of our analysis, extracted NFRs are traced to their implementations in the solution space [11]. The proposed approach exploits the textual semantic similarity between discovered NFRs and the textual content of code classes to identify code entities where such NFRs are potentially implemented [15]. The main assumption is that semantically-enabled retrieval methods can achieve higher accuracy levels by matching software artifacts based on their meanings rather than relying solely on lexical matching [16].

The remainder of this paper is organized as follows. Section II reviews seminal work in NFRs classification and traceability and motivates our research. Section III describes our procedure for extracting and classifying NFRs. Section IV describes a semantic method for tracing NFRs. Section V discusses threats to the study's validity. Finally, Section VI concludes the paper and describes directions for future work.

II. BACKGROUND AND MOTIVATION

This section briefly reviews the current state-of-the-art in non-functional requirements extraction, classification, and traceability and describes our main research motivations.

A. NFRs Classification

Cleland-Huang et al. [17] described an automated approach for classifying non-functional requirements present in various artifacts produced during the different phases of the software development process. The proposed approach is based on the main assumption that different types of NFRs can be distinguished by certain keywords known as indicator terms. Such terms are manually mined from existing software documents and used to train a classifier that can be later used to classify other requirements into different NFRs categories based on a function of the occurrence of indicator terms in the requirement's text. Evaluating the proposed approach over 30 student software projects showed a classification recall of 70%. However, a very high ratio of false positives was also reported.

Zhang et al. [18] conducted an empirical study to investigate the effectiveness of text mining techniques in NFRs classification. The authors analyzed the performance of three textual structures including N-grams, individual words, and multi-word phrases as representatives of various types of NFRs using Support Vector Machines with a linear kernel classifier. Repeating the experiment in [17] showed that individual words with Boolean weights were more effective in detecting and classifying NFRs. Results also showed that larger numbers of NFRs are required to enhance the inference accuracy.

Slankas and Williams [19] proposed an automated approach for extracting NFRs from several types of software documents such as requirements specifications, user manuals, and data agreements. A word vector representation is used to extract and classify sentences in such documents into 14 categories of NFRs. Individual sentences of each document are extracted, parsed, and classified using multiple classifiers, including K-NN, Sequential Minimum Optimizer (SMO), and Naive Bayes classifier. Results showed that SMO was the most effective in classifying requirements sentences.

Casamayor et al. [20] proposed a semi-supervised text categorization approach for detecting and classifying NFRs. The classifier is initially trained using a small set of manually categorized requirements. Using Expectation Maximization (EM), the classifier is then used to categorize unlabeled requirements. Users' feedback on classified requirements is also exploited to further enhance the accuracy. Empirical evaluation of the proposed approach showed higher levels of classification accuracy than fully supervised classification methods.

B. NFRs Traceability

Cleland-Huang and Schmelzer [21] suggested a design-based method to facilitate NFRs traceability through exploiting design patterns as intermediary objects. In particular, NFRs are initially elicited, analyzed, modeled, and mapped into appropriate design patterns. Traceability links are then manually

established between NFRs and code classes implementing their design patterns. Such links can be used to track and monitor changes that affect NFRs.

Mirakhorli and Cleland-Huang [10] conducted a comprehensive study to identify fundamental issues related to NFRs traceability. The authors analyzed tactical architectural decisions behind enforcing individual NFRs and investigated different techniques used to trace such decisions. The authors reported that NFRs are often difficult to trace as they are typically satisfied implicitly through non-documented design decisions. Furthermore, the authors discussed the benefits of tracing NFRs, including specific challenges that result from their cross-cutting nature and interdependencies.

Cleland-Huang et al. [22] proposed Goal Centric Traceability (GCT), a probabilistic approach to trace NFRs to individual code classes. In particular, NFRs and their interdependencies are modeled using a Softgoal Interdependency Graph (SIG). A probabilistic network model is then used to dynamically retrieve links between code classes and elements of the SIG. The main objective is to help system analysts manage the full impact of functional changes on different types of NFRs. The feasibility of the proposed approach was demonstrated through a case study. Results showed that using a probabilistic approach to automatically capture NFRs traceability links can save a considerable effort. However, manual evaluation of generated links is still required to filter out false positives.

Kassab et al. [9] proposed an approach to model functional and non-functional requirements and their interdependencies. The proposed approach exploits logic-based queries to identify and trace NFRs that are impacted by certain changes over the entire life cycle of a software project. A proof-of-concept study is conducted to illustrate the effectiveness of the proposed approach over a small Information Systems project. However, no empirical evidence was provided to demonstrate the feasibility of utilizing such an approach for larger systems.

C. Motivation

Our brief review shows that most current NFRs identification methods are supervised in the sense that a model has to be initially trained, using manually classified data, in order to be able to classify unseen-before instances. Such data might not always be available, especially that classifiers need large training datasets to achieve acceptable accuracy. Furthermore, experts from different application domains use different terminology to express their needs. Therefore, a classifier trained over a certain application domain might not necessarily work for other domains. In terms of traceability, our review shows that the majority of the proposed methods either assume the existence of explicitly defined NFRs that can be formalized and modeled, or rely on exploiting existing system architecture and design patterns to aid in the NFRs traceability process. While such methods can be effective for smaller systems, analyzing such structures and patterns can pose a scalability challenge when dealing with larger, and not very well-structured, systems [5], [10].

Motivated by these observations, in this paper we propose a novel approach for discovering and tracing NFRs. The proposed approach is unsupervised; no training of any sort is required. To enhance its practicality, the approach is independent, where NFRs are extracted and traced directly without any assumptions about any existing model or a well-defined system architecture. Furthermore, the proposed approach exhibits moderate complexity that allows it to scale to larger systems without exhausting time and space requirements.

III. NFRS EXTRACTION AND CLASSIFICATION

The approach proposed in this paper can be described as a multi-step procedure. Initially, individual words of requirements specifications are extracted and the pairwise semantic similarity between these words is calculated. Clustering is then used to group these words into cohesive clusters. These generated clusters are classified into different types of NFRs, mapped back to functional requirements, and traced to code classes implementing them. Fig. 1 depicts the proposed approach. Next is a description of each of these steps along with the associated research questions in greater detail.

A. Experimental Systems

Three software systems are used to conduct the analysis in this paper. These systems were chosen based on their size, the diversity of their application domains, and the availability of requirements documentation and original developers. In order to honor confidentiality agreements, we use the following pseudonyms to refer to these systems:

- *SmartTrip*: An Android mobile application that is designed to provide up-to-date routing and pricing information for users planning road trips through the United States. The application exploits several third-party APIs to make routing and accommodation recommendations.
- *SafeDrink*: A software system designed to help users to manage their drinking habits. The system has a mobile application interface which provides services such as recording the number of alcoholic beverages consumed and estimating the blood alcohol content (BAC). The system also provides a fast access to a list of emergency contacts and safety information.
- *BlueWallet*: A subscription-based web service that provides users with options to plan their budgets and manage their personal finances, including their credit cards, banking accounts, and tax information.

We start our analysis by indexing the requirements specifications of each system. Indexing is conducted by removing English stop-words (e.g., *the*, *was*) and performing stemming on remaining keywords. A dictionary-based conservative stemming approach is used to avoid over-stemming; where too much of the word is removed that the outcome of the stemming process is not a valid natural word (e.g., *general* and *generous* are stemmed to *gener*). Table I summarizes the outcome of the indexing process along with the characteristics of our different experimental systems, including the size of each system in terms of lines of source code (LOC), lines of code comments

TABLE I
EXPERIMENTAL SYSTEMS

System	LOC	CLOC	FR	WORDS	CLS.	DEVL.
<i>SmartTrip</i>	27.3k	7.7K	170	622	173	3
<i>SafeDrink</i>	44k	13.5K	214	717	266	3
<i>BlueWallet</i>	56.7k	19.4K	184	1034	374	2

(CLOC), number of functional requirements (FR), number of unique words extracted from each system’s requirements after indexing (WORDS), number of code classes (CLS), and number of developers available to participate in the study (DEVL).

B. Words Clustering

The cluster hypothesis implies that documents in the same cluster behave similarly with respect to relevance to information needs [23], [24]. This behavior can be tied to the information content of documents embedded in their individual words. In particular, words with similar meaning tend to be grouped in the same cluster [25]. We use these assumptions as basis for our approach. In particular, keywords extracted from the functional requirements of each system are clustered with the main objective of creating semantically coherent groups of natural language words. Formally, the main task at this step is to cluster a group of words $W = \{w_1, w_2, w_3, \dots, w_m\}$ into a set of disjoint groups of semantically similar terms $C = \{c_1, c_2, c_3, \dots, c_k\}$. Ideally, clusters in C should describe conceptual themes that pervade the original text. The main research questions at this step are, **RQ1**: *How to estimate the semantic distance between individual words?* and **RQ2**: *How to evaluate the performance of different clustering algorithms in creating meaningful groups of these words?*

C. Word Semantic Similarity

The main step in our approach is to calculate the pairwise semantic similarity between keywords extracted from requirements specifications. This step is crucial to the accuracy of the proposed approach as words’ pairwise similarity will determine how these words will be later clustered.

Several semantically-aware methods have been proposed in the literature to quantify similarity between natural language words. In our analysis, we adopt Normalized Google Distance (NGD) as the word semantic-similarity measure. Introduced by Cilibrasi and Vitányi [13], NGD is an information-theoretic method that uses Google counts to calculate semantic relatedness between words based on information distance and Kolmogorov complexity [26]. The main assumption is that the statistical co-occurrence of words (distributional cues) in the Web reflects their current similarity status in society, thus, can give an indication of their relatedness [27], [14], [28]. NGD has gained momentum in recent years due to its solid theoretical foundation, its simplicity, low computational complexity, and ability to achieve decent correlation levels with the human judgment of similarity [13]. Formally, to estimate the semantic distance between two words w_1 and w_2 using NGD, a Google search query Q is requested for

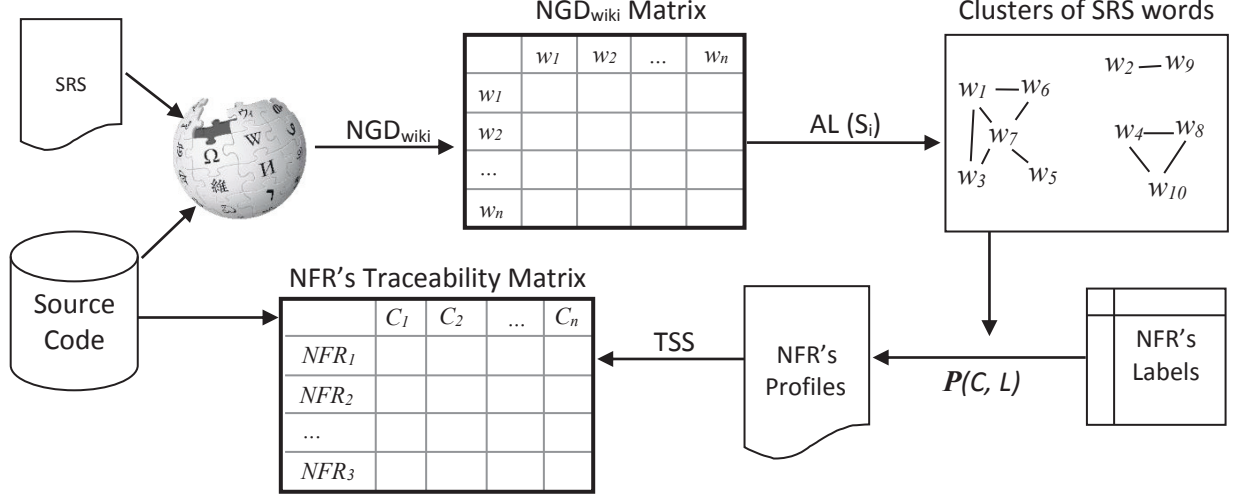


Fig. 1. NFRs extraction and traceability (NGD: Normalized Google Distance, AL: Average Linkage clustering algorithm, $s(i)$: Silhouette Coefficient, $P(C, L)$: Classification formula, TSS: Text Semantic Similarity, C_i : code class)

$Q(w_1)$, $Q(w_2)$, and $Q(w_1 \text{ AND } w_2)$. The semantic relatedness $NGD(w_1, w_2)$ is then measured as:

$$\frac{\max\{\log(D_1), \log(D_2)\} - \log(|D_1 \cap D_2|)}{\log(|D|) - \min\{\log(D_1), \log(D_2)\}} \quad (1)$$

where D_1 and D_2 are the hit counts of $Q(w_1)$ and $Q(w_2)$, or number of documents containing w_1 and w_2 respectively, and $|D_1 \cap D_2|$ is the hit count of $Q(w_1 \text{ AND } w_2)$, or number of Web documents that contain both w_1 and w_2 . The main tenet is that pages that contain both words indicate relatedness, while pages that contain only one of the words suggest the opposite. NGD is a distance measure, meaning that $NGD(w_1, w_1) = 0$, and the NGD of two words that do not occur together on any Web page, but do occur separately is ∞ . The value of NGD can be bounded between 0 and 1 using [29]:

$$nNGD = e^{-2NGD(w_1, w_2)} \quad (2)$$

The Google API is often used to build the hit-count matrix for NGD [30]. This API provides developers with an interface to integrate Google services in their programs. However, Google API enforces hard quotas on the number of free programmatic queries per time frame, allowing only a limited number of requests to be made. Another drawback of NGD, or any other method that relies on Web counts, stems from the fact that the Web is a constantly growing corpus and hit counts returned by search engines can vary in an unrepeatably manner [31], thus, raising major concerns about the reliability of such counts for similarity estimations [30]. In order to control for such effects, our word co-occurrence matrix is constructed from Wikipedia. Even though Wikipedia is naturally a smaller corpus than the entire Web, it is still the largest freely available, semi-structured, and constantly evolving source of textual human knowledge. Moreover, Wikipedia is often updated by experts, which makes it less noisy in comparison to the entire Web [32], [33]. To implement NGD using Wikipedia, the most

TABLE II
MOST SEMANTICALLY SIMILAR WORDS TO THE WORD `map` FROM *SmartTrip* AND THE WORD `credit` FROM *BlueWallet* USING NGD_{WIKI}

map		credit	
word	similarity	word	similarity
area	0.67	card	0.36
GPS	0.54	fee	0.35
zip	0.51	price	0.35
location	0.50	pin	0.34

recent XML dumps of English Wikipedia were downloaded. Individual articles were then extracted and indexed (removing English stop words and hyperlinks and performing stemming) and stored in a local database. SQL queries are used to construct the co-occurrence matrices of different experimental systems. $nNGD$ (Eq. 2) similarity is then calculated using these matrices. We refer to this approach as NGD_{WIKI} . Table II shows sample similarity calculations conducted using NGD_{WIKI} . The table shows the 4 most similar words to the word `map` from the *SmartTrip* system and the word `credit` from the *BlueWallet* system. These samples reflect how NGD_{WIKI} is able to detect sensible similarity relations that reflect the way these words are typically used by humans.

D. Clustering Algorithms

Several word clustering algorithms have been proposed in the literature [34], [35]. These algorithms apply several semantic schemes to group words into semantically coherent clusters. In our analysis, we adopt Hierarchical Agglomerative Clustering (HAC) as our main clustering mechanism. The underlying logic of agglomerative clustering is to successively merge the most similar elements in a data space into larger clusters. HAC clustering algorithms produce hierarchical structures that can be represented as dendrograms. A dendrogram

is a tree-structured graph that reflects how objects in a space are being grouped at different iterations of the algorithm [36].

The most popular variants of HAC include Complete Linkage (CL), Single Linkage (SL), and Average Linkage (AL). SL calculates the distance between two clusters as the distance between their most similar pair of elements, while CL uses the distance between most dissimilar pair of elements as the distance between two clusters, and AL merges the two clusters with the maximum average pairwise similarity between all their data elements. Formally, assuming the distance between any two data items in two clusters **A** and **B** is given by $d(a, b)$, the linkage (merging) criteria for clusters **A** and **B** can be described as follows:

- SL: $M(A, B) = \min\{d(a, b) : a \in A, b \in B\}$
- CL: $M(A, B) = \max\{d(a, b) : a \in A, b \in B\}$
- AL: $M(A, B) = \text{average}\{d(a, b) : a \in A, b \in B\}$

To answer our second research question (**RQ2**), we define the optimal clustering algorithm as the algorithm that maximizes the semantic coherence and distinctivity of the generated clusters. Semantic coherence can be described by cohesion, which is a measure of compactness, or how semantically related words in a cluster are. Distinctivity can be described by separation, or how semantically separated clusters are from one another. An optimal number of clusters should maximize coherence and distinctivity. To quantify such information, we adopt Silhouette Coefficient ($s(i)$) [37]. $s(i)$ is a robust measure that can be used to determine how relevant and distinctive a word to a cluster is. In particular, assuming a corpus has N words that have been divided into C clusters by a certain clustering algorithm, $s(i)$ of a certain word in a certain cluster can be calculated as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (3)$$

where $a(i)$ is the average distance between word i to other words in the same cluster, and $b(i)$ is the average distance between word i to words in its nearest neighbor cluster (minimum average distance between word i and other clusters). Distance is calculated as $1 - \text{Similarity}$, thus $\text{distance} = 1 - e^{-2\text{NGD}_{\text{Wiki}}(w_1, w_2)}$. Silhouette Coefficient can range from $[-1, 1]$, where a higher score relates to a model with better separation since $a(i) \ll b(i)$. In our analysis, we enforce a maximum size constraint of 40 words on the generated clusters [38]. This step is necessary to avoid the *chaining* effect in HAC clustering. In particular, if no size constraint is enforced on the generated clusters, HAC algorithms tend to create one big cluster that keeps growing during the clustering process and drags all items to it, or in some cases, very small clusters are iteratively created and suddenly grouped into one big cluster near the top of the hierarchy.

After the NGD_{Wiki} matrices are constructed for each system, words are clustered using the different HAC clustering algorithms and $s(i)$ is measured. Results (Fig. 2) show that Average Linkage (AL) algorithm is the most successful in extracting highly cohesive word clusters in all systems, achieving an average $s(i)$ of $\{0.63, 0.54, 0.47\}$ in *SmartTrip*, *SafeDrink*,

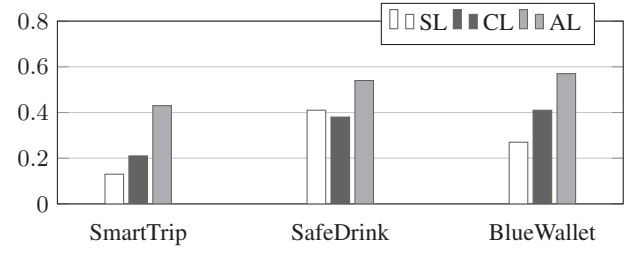


Fig. 2. Performance of different HAC clustering algorithms in terms of Silhouette Coefficient in *SmartTrip*, *SafeDrink*, and *BlueWallet*

TABLE III
NFRs CATEGORIES AND THEIR REPRESENTATIVE WORDS

NFR	Morphological Variants
<i>Security</i>	security, secure
<i>Performance</i>	performance, perform
<i>Accessibility</i>	access, accessible, accessibility
<i>Accuracy</i>	accuracy, accurately, accurate
<i>Portability</i>	portable, portability
<i>Safety</i>	safe, safety
<i>Legal</i>	legally, legal
<i>Privacy</i>	privacy, private
<i>Reliability</i>	reliable, reliability
<i>Availability</i>	availability, available
<i>Interoperability</i>	interoperable, interoperability

and *BlueWallet* respectively. CL and SL, on the other hand, failed to produce any meaningful clusters, with CL achieving slightly better results than SL. In general, AL tends to produce relatively balanced word clusters, where a word has to have a high average pairwise similarity to all other words in the cluster in order to be merged in, thus preserving the context, or the theme, of the cluster [39]. CL and SL, on the other hand, rely on a single word-pair to determine the distance between two clusters, which seems to be affecting the word clustering process negatively as other words that might be more distinctive to the topic may be ignored.

E. NFRs Classification

The main task at this step is to capture clusters of words that potentially represent non-functional requirements of the system. Formally, this task can be described as a standard classification problem where generated clusters are classified into NFRs and not NFRs (*other*) categories. To guide the classification process, a list of classification labels, describing basic types of NFRs is used. Even though there are almost 150 recognized categories of NFRs [2], at this stage of our analysis, only most general categories of NFRs that have been identified in the literature are used [19], [17]. These categories include: *security*, *performance*, *accessibility*, *accuracy*, *portability*, *safety*, *legal*, *privacy*, *reliability*, *availability*, and *interoperability*.

To assign word clusters to these NFRs labels, the average NGD_{Wiki} between each label and the words in each individual word cluster is calculated. To enhance the accuracy of the classification process, different morphological variants (Table III) of these labels are provided. An individual word's similarity

to a certain NFR is calculated as the maximum NGD_{Wiki} similarity between that word and any of the morphological variants of that NFR. Formally, assuming a cluster C , with N number of words $\{w_1, w_2, w_3, \dots, w_N\}$, and an NFR (L), with M number of morphological variants $\{l_1, l_2, l_3, \dots, l_M\}$, the probability that C belongs to L is given by:

$$P(C, L) = \frac{1}{N} \sum_{i=1}^N \max_{0 \leq j \leq M} NGD_{Wiki}(w_i, l_j), \quad (4)$$

After calculating the similarity of a certain word cluster to all NFRs, the cluster is assigned to the NFR with maximum similarity, given that the similarity exceeds a certain experimental threshold λ .

To measure the accuracy of the proposed approach, developers from different experimental systems were asked to manually classify the system's functional requirements into the different NFRs categories identified earlier (Table III), creating our ground-truth dataset. Each functional requirement can be classified under only one NFR. This experimental design decision was adopted to simplify the analysis at this stage of our research. In cases of conflicts (i.e., different developers classify a certain functional requirement under different NFRs) or vague cases (i.e., a functional requirement enforces multiple potential NFRs), a vote was taken. In case of a tie, the researcher participated in making the final decision. The different types of NFRs (Table III) were explained to the participants and the task was demonstrated by the researcher using sample functional requirements prior to the experiment. No time constraint was enforced.

The functional requirements of each system are then automatically classified into different NFRs categories based on the semantic similarity between the functional requirement and the clusters of words representing individual NFRs. More specifically, the textual semantic similarity measure (TSS) proposed by Mihalcea et al. [15] is used. TSS combines information from word-to-word similarity and word specificity to calculate the similarity between short paragraphs. Formally, the semantic similarity between two texts $sim(T_1, T_2)$ can be described as follows:

$$\frac{1}{2} \left(\frac{\sum (max(sim(w_i, T_2) \times idf(w_i)))}{\sum idf(w_i)} + \frac{\sum (max(sim(w_j, T_1) \times idf(w_j)))}{\sum idf(w_j)} \right) \quad (5)$$

For each word $w_i \in T_1$, the word in T_2 that has the highest semantic similarity to w_i is identified by $max(sim(w_i, T_2))$. $idf(w_i)$ is word w_i 's specificity. idf is calculated as the number of functional requirements in the system divided by the number of functional requirements that contain w_i . This process is repeated over all the words in T_1 and these values are summed. The summation is then divided by the summation of idf for all $w_i \in T_1$ for normalization. This process is repeated for T_2 , and the two values are averaged, giving a similarity score between T_1 and T_2 . This similarity measure

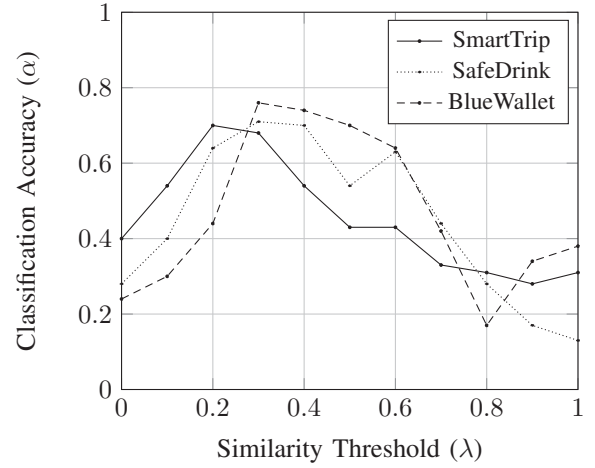


Fig. 3. Approach performance in terms of classification accuracy at different similarity thresholds

is used as a basis for classification. In particular, a functional requirement is assigned to the NFR with maximum TSS value, given that this value exceeds a similarity threshold λ .

In multi-class classification, the overall accuracy of the model (α) can be calculated as the number of correct classifications divided by the total number of classifications. This measure can be used as an objective function to determine optimal values of the similarity threshold λ that achieve the best results in each system. In particular, functional requirements are automatically classified using different values of λ , starting from 0, gradually increasing that value by 0.1 step. Classification results are compared with the ground-truth set at each step. Fig. 3 shows that the proposed approach achieved best classification performance at similarity threshold levels (λ, α) of $\{(0.2, 0.68), (0.3, 0.71), (0.3, 0.76)\}$ in *SmartTrip*, *SafeDrink*, and *BlueWallet* respectively.

To get more insights into these results, recall and precision values are calculated for each individual NFR in each system at average λ value of 0.27. Recall is calculated as the percentage of correctly classified functional requirements (true positives) from the set of functional requirements that enforce the NFR, and precision is calculated as the percentage of classified functional requirements under that NFR that are correct [19], [17]. Results are shown in Table IV which shows average (precision, recall) of (72%, 60%), (71%, 69%), (77%, 78%) for *SmartTrip*, *SafeDrink*, and *BlueWallet* respectively. In what follows, we discuss these results in greater detail.

F. Analysis

We start our results analysis by examining classification samples from different datasets. *SafeDrink* enforces a high *security* constraint. This NFR is characterized by the word cluster {password, login, email, authenticate, authorize, log, register}. These words appear in functional requirements such as “registered users should log-in using their email and password” and “the system shall authenticate users credentials”. In the *BlueWallet* system,

TABLE IV

NFRs CLASSIFICATION PERFORMANCE AT $\lambda = 0.28$. FR IS THE NUMBER OF FUNCTIONAL REQUIREMENTS CLASSIFIED MANUALLY UNDER THE ASSOCIATED NFRs CATEGORY, TP IS TRUE POSITIVE CLASSIFICATIONS, FP IS FALSE POSITIVES, P IS PRECISION, AND R IS RECALL

	SmartTrip					SafeDrink					BlueWallet				
NFR	FR	TP	FP	R	P	FR	TP	FP	R	P	FR	TP	FP	R	P
<i>security</i>	1	1	1	1	0.5	23	23	7	1	0.77	24	24	4	1	0.86
<i>performance</i>	29	24	9	0.83	0.73	9	8	9	0.89	0.47	8	8	8	1	0.5
<i>accessibility</i>	3	3	1	1	0.75	32	28	6	0.88	0.82	6	3	2	0.5	0.6
<i>accuracy</i>	7	4	4	0.57	0.5	11	7	2	0.64	0.78	34	30	9	0.88	0.77
<i>portability</i>	14	11	6	0.79	0.65	21	18	11	0.86	0.62	2	2	0	1	1
<i>safety</i>	10	10	3	1	0.77	21	17	4	0.81	0.81	0	0	0	NA	NA
<i>legal</i>	4	2	4	0.5	0.33	38	33	9	0.87	0.79	14	6	4	0.43	0.6
<i>privacy</i>	3	2	2	0.67	0.5	12	10	3	0.83	0.77	9	9	4	1	0.69
<i>reliability</i>	20	16	6	0.8	0.73	4	3	6	0.75	0.33	14	11	3	0.79	0.79
<i>availability</i>	7	4	3	0.57	0.57	NA	0	0	NA	NA	4	2	3	0.5	0.4
<i>interoperability</i>	19	4	6	0.21	0.4	14	2	0	0.14	1	0	0	0	NA	NA
<i>other</i>	53	39	6	0.72	0.86	29	4	4	0.14	0.5	69	45	7	0.65	0.87
average				0.72	0.6				0.71	0.69				0.77	0.78

the *security* constraint is emphasized through the sophisticated encryption algorithm the system uses to protect the user’s data. In particular, this system deals with sensitive user information. It requires encryption of all data stored on the user’s mobile device or exchanged over the network. Therefore, the *security* constraint in this system is enforced through the word group {encrypt, decrepit, biometric, sensitive, restrict, prevent, deny, attack, malicious}. Such words appear in functional requirements such as “*three keys encrypt the data with a 168-bit encryption*”. In addition to the security constraint, *BlueWallet* also enforces an *accuracy* constraint, captured by the words {precision, decimal, margin, tolerance, measurement, calculation, exact}. Such words appear in functional sentences such as “*the precision of the calculation is controlled by the predefined epsilon*” and “*total should be rounded to two decimal places*”.

SafeDrink maintains a list of different states’ drinking laws and provides information for local law enforcement agencies and current drinking regulations in the area, thus enforcing a *legal* constraint. This NFR is captured by the cluster {law, lawyer, regulation, guidelines, legal, police, insurance, comply, standard, ramification, liability, state}. *SafeDrink* also enforces an *accessibility* constraint. In particular, the target users of this application are people under the influence of alcoholic substances. Therefore, unlike *SmartTrip*, which assumes that users are not operating a vehicle under the influence, *SafeDrink* provides extra accessibility requirements. This constraint is detected through the word cluster {easy, access, thump, font, magnify, picture, color, graphic, command, red, blue, green, cup, access, look, feel, simple, appealing}.

We also observe that *SmartTrip* does not enforce a *security* constraint. The system simply relies on the host’s security

measures to protect its data. However, it enforces a *performance* constraint as it performs several real-time operations such as locating all nearby hotels and using different API’s to get routing and pricing information. This constraint is captured through the word cluster {processor, speed, memory, response, time, date, start-up, millisecond, second, hour, trans, transmit, signal, live}.

In terms of limitations, the proposed approach was not very successful in detecting certain cases. For instance, *SmartTrip* and *safeDrink* enforce an *interoperability* constraint. This NFR ensures that all integrated systems can work together and exchange information, and is usually enforced by functional requirements that manage the interaction between the client side interface, the server, and the third-party API’s. We notice that the clustering algorithm failed to group keywords with semantic similarity to the word *interoperability*. Two reasons can be identified for this problem. First, the word *Interoperability* appears on a relatively smaller number of Wikipedia articles (63 articles). Therefore, its chances to co-occur with other words are limited. Furthermore, words that are often associated with this particular NFR tend to be scattered over the clustering space, and often grouped with clusters related to performance and portability. Another example of potential failure is polysemy, or when the word has a different meaning based on the context. For instance, in *BlueWallet*, the word *fault* appears in two contexts, in one instance referring to a software fault, and in another instance referring to a mathematical fault. Therefore, it could either be an indicator of *reliability* or *accuracy*. Semantically, the word *fault* stands at almost the same NGD_{Wiki} distance from these two NFRs, where $NGD(accuracy, fault) = 0.25$ and $NGD(reliability, fault) = 0.24$. This problem can be probably controlled by expanding the window of co-occurrence to deal with phrases rather than single words. However, such claims are yet to be evaluated in our future work.

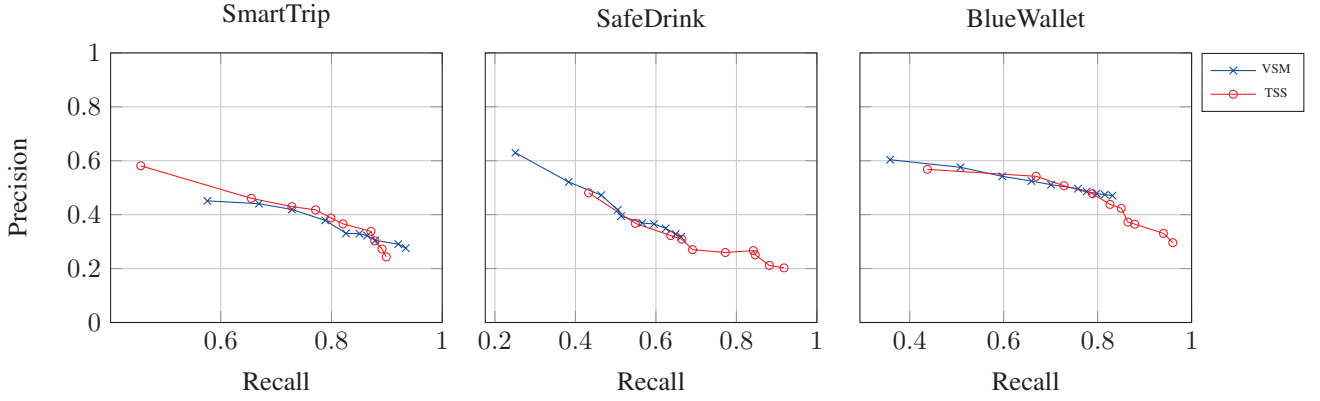


Fig. 4. Performance of VSM and TSS in terms of Recall and Precision at different threshold points

IV. NFR'S TRACEABILITY

In the second phase of our analysis, we trace discovered NFRs to their implementations in the solution space. NFRs traceability links help to verify that all quality constraints in the system have been implemented, aid in tasks such as regression testing, and support change impact analysis [7], [10], [22]. The main research question at this step is **RQ4: How to establish traceability links between discovered NFRs and source code entities?**

Modern requirements traceability tools exploit Information Retrieval (IR) methods for automated support [16], [40]. The main assumption is that software artifacts sharing similar vocabulary are likely to be traceable from one another. In our analysis, we use TSS (Eq. 5) [15] as the underlying IR method to match NFRs with code entities. The decision to adopt TSS is based on previous findings in IR-based traceability, which indicate that relying solely on lexical matching often leads to information loss [16], [40]. This can be explained based on the fact that, as projects evolve, new and inconsistent terminology gradually finds its way into the system's taxonomy, causing various types of related system artifacts (requirements, source code, design cases) to exhibit a large degree of variance in their lexical contents [41]. This phenomena, known as the *vocabulary mismatch* problem, is regarded as one of the principal causes of poor accuracy in traceability engines [40]. Therefore, by matching words based on their meaning, as well as their lexical structure, TSS is expected to help to bridge the textual gap between different system's artifacts, thus improve the retrieval accuracy. In our analysis we work at class granularity level, where individual NFRs are matched with code classes potentially implementing them [21]. The decision to consider class level is based on the observation that, given the sparsity of text in source code, higher granularity levels (e.g., a code snippet or a method), do not provide sufficient information for a method such as TSS to work [27]. On the other hand, at lower levels (e.g., package), the window size is so big that words' distributional cues become meaningless,

thus, increasing the likelihood of irrelevant and misleading information and depriving TSS of context.

A. Traceability Link Retrieval

Initially, regular expressions are used to analyze code segments and extract different text patterns (method signatures, variables, comments, and string literals). Extracted code identifiers are split into their constituent words based on standard camel-casing (e.g., UserID is split to User and ID). Programming and English stop-words are then filtered out (e.g., int, public). The outcome of the indexing process is compact content descriptors, or *profiles* containing keywords of individual code classes. Individual natural language words from these profiles are tabulated in the system's words co-occurrence matrix based on their Wikipedia co-occurrence patterns with the requirements words. TSS is then used match code classes with NFRs, where specificity ($idf(w_i)$) is calculated considering source code classes as system artifacts.

To get a sense of TSS performance, we compare its accuracy with Vector Space Model (VSM). VSM with $tf.idf$ weights is often used in traceability studies as an experimental baseline that relies on lexical matching of software artifacts [16], [40]. For each experimental system, discovered NFRs clusters are matched with all code classes in the system using TSS and VSM. Classes are then ranked in a list of candidate traceability links based on their similarity to the NFR being traced. The performance of TSS and VSM is measured in terms of precision (P) and recall (R). Recall measures coverage and is defined as the percentage of correct links that are retrieved, and precision measures accuracy and is defined as the percentage of retrieved links that are correct. Answer-sets were prepared and provided by original developers of each system. In particular, developers of different systems were asked to identify individual code classes implementing different NFRs discovered earlier. Performance is reported over multiple threshold levels $\{0.1, 0.2, \dots, 1\}$ (i.e., at threshold 0.1 only top 10% of the retrieved classes is considered).

Fig. 4 shows the performance of VSM and TSS in the three experimental systems. Results show that, VSM achieves an average precision of 35% in all three systems in comparison

to TSS, which achieves a 24% average precision. TSS, on the other hand, is able to achieve higher recall levels at lower threshold values, keeping up the good performance until achieving a 93% average recall, while VSM stops at a 81% average recall. However, TSS precision at these higher threshold levels is lower due the increase in the number of false positives. The high recall levels of TSS can be explained based on its ability to capture a large percentage of equivalent words (synonyms) in the system. Previous investigations in IR-based traceability show that using a domain-specific synonymy thesaurus to aid in the retrieval process often leads to higher recall levels [16], [40]. TSS simulates this process by considering the semantic similarity of words, captured through NGD_{Wiki} , thus provides an automated alternative to manually maintaining a system thesaurus, a process that is often described as time consuming and error-prone.

The relatively low precision in VSM and TSS can be explained based on the impurity of NFRs words clusters. In particular, due to the near-optimal clustering process, some NFRs clusters contain noise, or unrelated words, that contribute to the similarity calculations, thus enhance the possibility of retrieving unrelated code classes. However, given that recall is often emphasized over precision in traceability research [40], the proposed approach can be considered effective for NFRs traceability link recovery, achieving performance levels that can be adequate for practical applications [16], [40].

B. Complexity and Scalability

In terms of space complexity, the proposed approach, shown in Fig. 1, works with a main matrix that contains raw co-occurrence data extracted from Wikiepdia, such data is used to generate the NGD_{Wiki} similarity matrix. These two matrices are symmetrical. This reduces the memory space requirements to $O(n(n+1)/2)$. The main space concern associated with the proposed approach is the storage required to store the entire Wikipedia on disk. In terms of time complexity, once the co-occurrence matrix is established, the bottleneck step in the proposed approach is clustering, given that the complexity of HAC clustering algorithms is $O(n^2)$, where n is the number of data items in the space. In terms of operational complexity, the proposed approach is fully unsupervised, no data is needed ahead to train the model. Such moderate space and time complexities, as well as the simplicity of the proposed approach, enhance its scalability to larger systems. In particular, even though the co-occurrence matrix has to be updated to accommodate more words, no additional processing is required other than a straightforward tabulation of new co-occurrence information.

V. EXPERIMENTAL LIMITATIONS

The experiment presented in this paper has several limitations that might affect the external validity of the study. In particular, the results of the experiment might not generalize beyond the specific experimental settings used in this paper. A potential threat to the external validity stems from the systems used to conduct the experiment. Our experimental set consisted

of only three midsize software systems with a limited number of functional requirements. Such systems are likely to exhibit different characteristics from large scale systems or systems from different application domains. However, we believe that using three software systems from different application domains helps to mitigate this threat.

Certain design decisions might also impact the external validity of our results. For example, we only experimented with the family of HAC clustering algorithms. It is still unclear whether other clustering algorithms (e.g., K-means) might be more appropriate for our approach. Other threat might stem from the fact that only a limited number of NFRs (Table III) were included in the analysis. However, these NFRs are the most general types that are often considered in related research [19], [17]. Moreover, considering all categories and subcategories of NFRs will substantially increase the complexity of the study and the time required for our subjects to classify the system's functional requirements into such a large number of NFRs categories, thus impacting the overall accuracy of the analysis.

In terms of traceability, only TSS and VSM were considered in our analysis. Other IR methods such as Latent Semantic Indexing (LSI) or Latent Dirichlet Allocation (LDA) might achieve different results [16]. Nonetheless, the way our approach is designed (Fig. 1) enables other researchers to repeat our experiment under different settings, to maybe achieve better results, or tune the approach to work with other systems.

Internal validity refers to factors that might affect the causal relations established in the experiment. A potential threat to the proposed study's internal validity is the fact that human judgment is used to prepare our classification (Sec. III-E) and traceability (Sec. IV) answer-sets. This might result in an experimental bias as different project personnel might have different understanding of the system, its constraints and traceability relations. While these threats are inevitable, they can be partially mitigated by using multiple human subjects to classify each requirement. Finally, the class-granularity level adopted in our analysis can also influence the performance. In particular, different granularity levels might considerably change the behavior of co-occurrence based methods.

VI. CONCLUSIONS

This paper proposes a novel, unsupervised, and scalable approach for extracting, classifying, and tracing non-functional requirements of software systems. The analysis in this paper is based on the main assumption that NFRs tend to be implicitly enforced by the functional requirements of the system. In particular, the proposed approach exploits co-occurrence statistics in massive textual sources of human knowledge to estimate semantic similarity between keywords extracted from requirements specifications. These keywords are clustered using Average Linkage clustering algorithm into semantically coherent groups that are later classified into different categories of NFRs, mapped to functional requirements, and traced to their implementing code classes. Three software systems were used to conduct our experimental analysis. Results show

that the proposed approach managed to discover and classify NFRs with an average classification accuracy of 73%. In terms of traceability, results showed that the semantically-enabled IR method TSS (Eq. 5) managed to retrieve NFR-to-code traceability links with an average recall of 93% at an average precision of 25%, outperforming lexical-based VSM in terms of recall and providing an automated alternative for building a system synonymy thesaurus to aid in the retrieval process.

Two research directions will be pursued in our future work. First, more empirical studies will be conducted using larger systems to develop a better understanding of different aspects of the proposed approach, including its scalability, usability, and scope of applicability. Secondly, we will continue to exploit the semantics of Wikipedia to create a semantically-aware framework that not only discovers basic types of NFRs, but also aids in their modeling and analysis by mining interdependencies and trade-offs between discovered NFRs.

REFERENCES

- [1] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. John Wiley and Sons, 1998.
- [2] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-functional Requirements in Software Engineering*. Kluwer Academic, Boston, MA, 2000.
- [3] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, 2001.
- [4] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, 2009, vol. 5600, pp. 363–379.
- [5] D. Gross and E. Yu, "From non-functional requirements to design through patterns," *Requirements Engineering*, vol. 6, no. 1, pp. 18–36, 2000.
- [6] M. Glinz, "On non-functional requirements," in *IEEE International Requirements Engineering Conference*, 2007, pp. 21–26.
- [7] O. Gotel, J. Cleland-Huang, J. Huffman-Hayes, A. Zisman, A. Egyed, P. Grnbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, "The grand challenge of traceability (v1.0)," in *Software and Systems Traceability*. Springer London, 2012.
- [8] L. C. John Mylopoulos and B. Nixon, "Representing and using nonfunctional requirements: a process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.
- [9] M. Kassab, O. Ormandjieva, and M. Daneva, "A metamodel for tracing non-functional requirements," in *World Congress on Computer Science and Information Engineering*, 2009, pp. 687–694.
- [10] M. Mirakhorli and J. Cleland-Huang, "Tracing non-functional requirements," in *Software and Systems Traceability*, A. Zisman, J. Cleland-Huang, and O. Gotel, Eds. Springer-Verlag, 2012, pp. 299–320.
- [11] J. Cleland-Huang, C. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.
- [12] L. M. Cysneiros, "Evaluating the effectiveness of using catalogues to elicit nonfunctional requirements," in *Workshop em Engenharia de Requisitos*, 2007, pp. 107–115.
- [13] R. Cilibrasi and P. Vitanyi, "The google similarity distance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 370–383, 2007.
- [14] K. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Computer Linguistics*, vol. 16, no. 1, pp. 22–29, 1990.
- [15] R. Mihalcea, C. Corley, and C. Strapparava, "Corpus-based and knowledge-based measures of text semantic similarity," in *National Conference on Artificial Intelligence*, 2006, pp. 775–780.
- [16] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requirements Engineering*, pp. 1–20, 2014.
- [17] J. Cleland-Huang, R. Settini, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, 2007.
- [18] W. Zhang, Y. Yang, Q. Wang, and F. Shu, "An empirical study on classification of non-functional requirements," in *International Conference on Software Engineering and Knowledge Engineering*, 2011, pp. 190–195.
- [19] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, 2013, pp. 9–16.
- [20] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.
- [21] J. Cleland-Huang and D. Schmelzer, "Dynamically tracing non-functional requirements through design pattern invariants," in *Workshop on Traceability in Emerging Forms of Software Tracing Non-Functional Requirements*, 2003.
- [22] J. Cleland-Huang, R. Settini, O. BenKhadra, E. Berezanskaya, and S. Christina, "Goal-centric traceability for managing non-functional requirements," in *International Conference on Software Engineering*, 2005, pp. 362–371.
- [23] C. J. van Rijsbergen, *Information Retrieval*. Butterworths, 1979.
- [24] N. Niu and A. Mahmoud, "Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited," in *IEEE International Requirements Engineering Conference*, 2012, pp. 81–90.
- [25] M. Hearst and J. Pedersen, "Reexamining the cluster hypothesis: scatter/gather on retrieval results," in *international ACM SIGIR conference on Research and development in information retrieval*, 1996, pp. 76–84.
- [26] M. Li, X. Chen, X. Li, B. Ma, and Vitanyi, "The similarity metric," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.
- [27] J. Bullinaria and J. Levy, "Extracting semantic representations from word co-occurrence statistics: A computational study," *Behavior Research Methods*, vol. 39, no. 3, pp. 510–526, 2007.
- [28] K. Lund and C. Burgess, "Producing high-dimensional semantic spaces from lexical co-occurrence," *Behavior Research Methods, Instruments, Computers*, vol. 28, no. 2, pp. 203–208, 1996.
- [29] J. Gracia, R. Trillo, M. Espinoza, and E. Mena, "Querying the web: a multontology disambiguation method," in *International Conference on Web Engineering*, 2006, pp. 241–248.
- [30] T. Funahashi and H. Yamana, "Reliability verification of search engines' hit counts: How to select a reliable hit count for a query," in *International Conference on Current Trends in Web Engineering*, 2010, pp. 114–125.
- [31] M. Thelwall, "Extracting accurate and complete results from search engines: Case study windows live," *Journal of the American Society for Information Science and Technology*, vol. 59, no. 1, pp. 38–50, 2008.
- [32] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis," in *International Joint Conference on Artificial Intelligence*, 2007, pp. 1606–1611.
- [33] M. Strube and S. Ponzetto, "Wikirelate! computing semantic relatedness using wikipedia," in *National Conference on Artificial Intelligence*, 2006, pp. 1419–1424.
- [34] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter, "Distributional word clusters vs. words for text categorization," *Journal of Machine Learning Research*, vol. 3, pp. 1183–1208, 2003.
- [35] N. Slonim and N. Tishby, "Document clustering using word clusters via the information bottleneck method," in *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2000, pp. 208–215.
- [36] N. Anquetil, C. Fourrier, and T. Lethbridge, "Experiments with clustering as a software remodularization method," in *Working Conference on Reverse Engineering*, 1999, pp. 235–255.
- [37] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, no. 1, pp. 53–65, 1987.
- [38] J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei, *Reading tea leaves: How humans interpret topic models*. Curran Associates, 2009, pp. 288–296.
- [39] C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining Text Data*. Springer US, 2012, pp. 77–128.
- [40] J. Huffman-Hayes, A. Dekhtyar, and S. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.
- [41] G. Antoniol, Y. Gueheneuc, E. Merlo, and P. Tonella, "Mining the lexicon used by programmers during software evolution," in *International Conference on Software Maintenance*, 2007, pp. 14–23.