Requirements Eng (2016) 21:357–381
DOI 10.1007/s00766-016-0252-8

RE 2015

Detecting, classifying, and tracing non-functional software requirements

Anas Mahmoud[1] · **Grant Williams**[1]

Received: 12 November 2015 / Accepted: 28 April 2016 / Published online: 4 May 2016

Abstract In this paper, we describe a novel unsupervised approach for detecting, classifying, and tracing non-functional software requirements (NFRs). The proposed approach exploits the textual semantics of software functional requirements (FRs) to infer potential quality constraints enforced in the system. In particular, we conduct a systematic analysis of a series of word similarity methods and clustering techniques to generate semantically cohesive clusters of FR words. These clusters are classified into various categories of NFRs based on their semantic similarity to basic NFR labels. Discovered NFRs are then traced to their implementation in the solution space based on their textual semantic similarity to source code artifacts. Three software systems are used to conduct the experimental analysis in this paper. The results show that methods that exploit massive sources of textual human knowledge are more accurate in capturing and modeling the notion of similarity between FR words in a software system. Results also show that hierarchical clustering algorithms are more capable of generating thematic word clusters than partitioning clustering techniques. In terms of performance, our analysis indicates that the proposed approach can discover, classify, and trace NFRs with accuracy levels that can be adequate for practical applications.

Keywords Classification · Non-functional requirements · Information retrieval · Semantics

✉ Anas Mahmoud
mahmoud@csc.lsu.edu

[1] Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA 70803, USA

1 Introduction

Requirements engineering (RE) is a human-centric process. Software functional requirements (FRs) are products of stakeholders' knowledge of their application domain. Such knowledge, expressed mainly in natural language (NL) [36, 56], propagates throughout the entire life cycle of the software project, from early RE through requirements description, to the solution space through code identifiers and internal code comments [4, 30]. The ability to capture and model such semantic knowledge has been proved vital for providing automated solutions for various basic RE tasks, such as requirements elicitation [58], analysis [77], traceability [60], and reuse [12].

Following this line of research, in this paper we exploit the textual semantics of software FRs to discover the non-functional requirements (NFRs) of the system. NFRs describe a set of quality attributes that a software system should exhibit [48]. Such attributes enforce operational constraints on different aspects of the system's behavior, such as its usability, security, reliability, performance, and portability [17]. Explicitly identifying NFRs early in the software process is critical for making initial design decisions and later for evaluating architectural alternatives for the system [68]. 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 [16]. Part of this phenomenon can be attributed to the vague understanding of what NFRs actually are and the lack of effective NFR elicitation, modeling, and documentation methods [35, 37, 39].

Motivated by these observations, we propose a novel, unsupervised, and computationally efficient approach for discovering and tracing NFRs in software systems. The

main assumption is that, even if they are not explicitly defined, NFRs tend to be embedded in the functional requirements of the system [25]. For instance, a system with a *login* feature is assumed to be secure. However, such information is typically scattered over the system's requirement 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.

Our earlier work in this domain has appeared in [59]. In this paper, we extend this work by conducting a comprehensive systematic analysis of a series of semantic methods, commonly used in NLP, to estimate the semantic similarity between words extracted from the functional requirements of software systems. These methods include: latent semantic analysis (LSA) [29], co-occurrence methods, including Normalized Google Distance (NGD) [19] and pointwise mutual information (PMI) [84], and thesaurus-based methods, including techniques that utilize the linguistic database WordNet [52]. Our main objective is to identify the most effective semantic schemes in capturing the notion of similarity between natural language words appearing in software FRs. Such information is then used for clustering these words into thematic groups. In particular, several partitioning and hierarchical clustering algorithms are tuned through a semantically aware objective function to identify cluster configurations that produce the most semantically cohesive clusters of FR words. These clusters are classified into different categories of NFRs based on their semantic similarity to basic NFR labels. Extracted NFR clusters are then traced to their implementation in the solution space based on their semantic similarity to the textual information extracted from source code. The proposed approach is calibrated and evaluated using three software systems from different application domains. In summary, the contributions of this extension can be described as follows:

- A systematic evaluation of the accuracy of a series of word semantic similarity methods and clustering techniques in the context of software requirements.
- A semantically aware cluster quality measure designed to identify cluster configurations that generate the most semantically coherent clusters of FR words.
- A computationally efficient technique for extracting the natural language content of source code to support NFR traceability link recovery.

The remainder of this paper is organized as follows. Section 2 reviews seminal work in NFR classification and traceability and motivates our research. Section 3 presents the set of semantic similarity methods and clustering techniques used in our investigation, and Sect. 4 describes the implementation, analysis, and evaluation of these methods and techniques. Section 5 describes the procedure used to detect different NFRs in the system. Section 6 assesses the performance of different IR methods in tracing detected NFRs to their implementation. Section 7 discusses the main threats to the study's validity. Finally, Sect. 8 concludes the paper and discusses prospects of future work.

2 Background and motivation

Due to their pervasive nature, and the lack of robust modeling and documentation techniques, NFRs are often overlooked during the requirements elicitation phase. Failure to identify NFRs early in the process impacts the overall quality of the system [39, 64]. In particular, NFRs tend to be connected through various interdependencies and trade-offs that span over multiple modules in the solution space [46]. 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 architectural anomalies and erosion problems [63, 68]. Furthermore, the lack of explicitly defined NFRs complicates the process of mapping, or tracing, such high-level quality constraints to their low-level implementation [20, 25, 37]. The availability of such information becomes vitally important in safety critical systems. Failure to satisfy the quality constraints in such systems can lead to catastrophic consequences [21, 70].

These realizations have motivated researchers to look for automated methods to enable early discovery and classification of NFRs and later to trace these NFRs to their implementation. In what follows, we review the current state of the art in NFR classification and traceability research, describe our main research motivations, and lay out the skeleton of our approach.

2.1 Related work

The analysis in this paper can be divided into two main phases. The first phase is concerned with detecting and classifying individual NFRs in software systems, and the second phase is concerned with tracing these NFRs to their implementation. In what follows, we briefly review seminal work related to each of these two phases.

2.1.1 NFR detection and classification

Cleland-Huang et al. [24] described an automated approach for classifying NFRs present in various software artifacts produced during the life cycle of the software project. The proposed approach was based on the assumption that different types of NFRs can be distinguished by certain

keywords known as the indicator terms. In particular, a set of correctly pre-classified functional requirements was used to train a classifier to automatically discover and assess the terms that indicated different types of NFRs. The generated classifier was then used to classify unseen-before functional requirements into different NFR categories based on a function of the occurrence of indicator terms in these requirements. An industrial case study was conducted to evaluate the proposed approach. The results showed an average classification recall of 80 %. However, a very high ratio of false positives (21 % precision) was also reported.

Zhang et al. [88] conducted an empirical study to investigate the effectiveness of text mining techniques in NFR classification. Using support vector machines (VSM) with a linear kernel classifier, the authors analyzed the performance of three textual structures including N-grams and individual words and phrases as representatives of various types of NFRs. Repeating the experiment in [24] showed that words with Boolean weights were more effective in detecting and classifying NFRs. Results also showed that larger numbers of NFRs were needed to enhance the inference accuracy.

Slankas and Williams [78] proposed an automated approach for extracting NFRs from several types of software documents such as requirement specifications, user manuals, and data agreements. A word vector representation was used to extract and classify sentences in such documents into 14 categories of NFRs. Individual sentences of each document were extracted, parsed, and classified using multiple classifiers, including K-NN, Sequential Minimum Optimizer (SMO), and naive Bayes classifier. The results showed that SMO was the most effective in classifying requirements' sentences.

Casamayor et al. [13] proposed a semi-supervised text categorization approach for detecting and classifying NFRs. The classifier was initially trained using a small set of manually categorized requirements. Using Expectation Maximization (EM), the classifier was then used to categorize unlabeled requirements. Users' feedback on classified requirements was also exploited to further enhance the accuracy. Empirical evaluation of the proposed approach showed that it achieved higher levels of classification accuracy than fully supervised classification methods.

2.1.2 NFR traceability

Cleland-Huang and Schmelzer [22] suggested a design-based method to facilitate NFR traceability through exploiting design patterns as intermediary objects. In particular, NFRs were initially elicited, analyzed, modeled, and mapped into appropriate design patterns. Traceability links were then manually established between NFRs and code classes implementing their design patterns. Such links were used to track and monitor changes that affect NFRs.

Mirakhorli and Cleland-Haung [63] conducted a comprehensive study to identify fundamental issues related to NFR traceability. The authors analyzed tactical architectural decisions behind enforcing individual NFRs and investigated different techniques used to trace such decisions. Results showed that NFRs were often difficult to trace as they were typically satisfied implicitly through non-documented design decisions. Furthermore, the authors discussed the benefits of tracing NFRs, including specific challenges related to their cross-cutting nature and interdependencies.

Cleland-Huang et al. [23] proposed goal-centric traceability (GCT), a probabilistic approach to trace NFRs to individual code classes. In particular, NFRs and their interdependencies were modeled using a Softgoal Interdependency Graph (SIG). A probabilistic network model was then used to dynamically retrieve links between code classes and elements of the SIG. The main objective was to help system analysts to manage the full impact of functional changes on different types of NFRs. The feasibility of the proposed approach was demonstrated through a case study. The results showed that using a probabilistic approach to automatically capture NFR traceability links saved a considerable effort. However, manual evaluation of generated links was still required to filter out false positives.

Kassab et al. [46] proposed an approach to model functional and non-functional requirements and their interdependencies. The proposed approach exploited logic-based queries to identify and trace NFRs that were impacted by certain changes over the entire life cycle of a software project. A proof-of-concept study was 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.

Frati et al. [70] suggested a method for representing and tracing basic NFRs in safety critical systems. The authors used formal models to demonstrate the satisfaction of a specific NFR requirement in the system through certain predefined relations. Such relations were then used to establish traceability links between design artifacts and requirements and link requirements to validation and verification elements (test cases) in the system.

2.2 Motivation, approach, and research questions

Our brief review shows that most current NFR detection and classification 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 since classifiers need large training datasets to achieve acceptable accuracy. Furthermore, experts from different application domains use different terminologies to express their needs. Therefore, a classifier trained over a certain application domain might not necessarily work for other domains [24].

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 NFR 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, or ill-structured, systems [39, 63].

Motivated by these observations, in this paper we describe a novel approach for detecting, classifying, and tracing NFRs. The proposed approach can be described as a multi-step procedure. Initially, individual words of requirement 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. Figure 1 depicts our approach. This 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 an existing model or a well-defined system architecture. Furthermore, the proposed approach exhibits moderate computational complexity that allows it to scale to larger systems without exhausting time and space requirements.

In what follows, we describe in detail the main steps of approach in Fig. 1, including the main theoretical assumptions and experimental analysis associated with each step. In particular, we attempt to answer the following research questions:

- **RQ1:** *What is the most accurate measure of semantic similarity between FR words?*
- **RQ2:** *What is the most effective clustering algorithm in generating meaningful clusters of FR words?*
- **RQ3:** *How can the generated NFRs be traced to source code artifacts?*

3 NFR classification

The first phase of our analysis is concerned with detecting potential NFRs enforced in the system. Our research assumptions at this phase are based on the main assumptions of the cluster hypothesis. This hypothesis states that documents in the same cluster behave similarly with respect to relevance to information needs [11, 67]. 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 [41]. We use these assumptions as the basis for our approach. In particular, keywords extracted from the functional requirements of the software 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 $W = \{w_1, w_2, w_3, \ldots, w_m\}$ into a set of disjoint groups of semantically similar words $C = \{c_1, c_2, c_3, \ldots, c_k\}$. Ideally, clusters in C should describe conceptual themes that

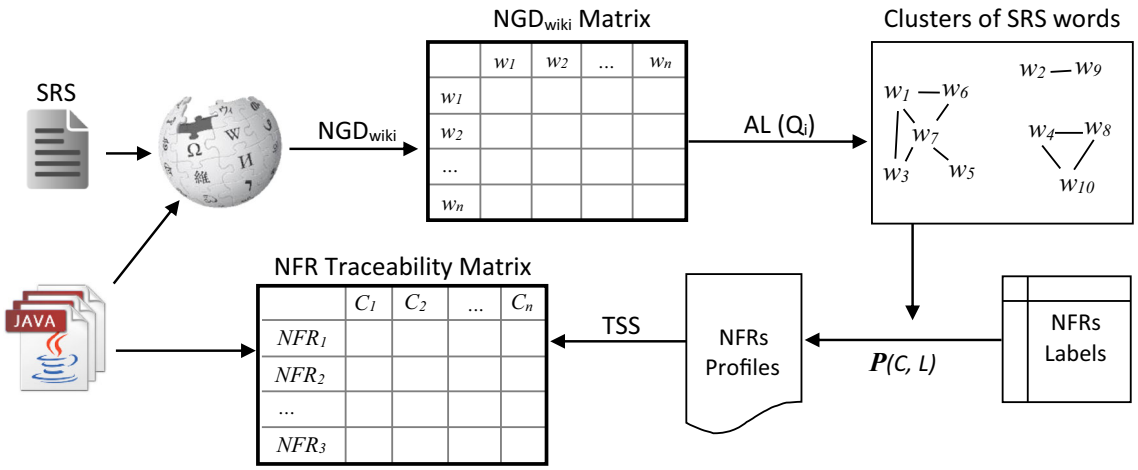

Fig. 1 NFR extraction and traceability (*NGD* Normalized Google Distance, *AL* average linkage clustering algorithm, Q_i cluster quality objective function, $P(C, L)$ classification formula, *TSS* text semantic similarity, C_i code class)

pervade the original text. Such themes might potentially represent quality constraints enforced in the system.

Identifying optimal cluster configurations that best fit a certain data space can be an NP-hard problem. However, near-optimal solutions can be determined experimentally. To determine such configurations, we investigate the performance of a series of semantic similarity measures and word clustering techniques to identify the most effective configurations that can produce semantically coherent, or thematic, clusters of FR words (i.e., **RQ1** and **RQ2**). Next is a description of these methods and techniques in greater detail.

3.1 Semantic similarity of FR words

Several methods have been proposed in the literature to estimate semantic similarity between natural language words. Such methods exploit a broad spectrum of semantic schemes, at different levels of computational complexity, to capture association relations between words in a corpus. To answer **RQ1**, in this paper we consider three categories of similarity methods that have been intensively used in NLP research. These methods include: latent semantic analysis, co-occurrence based methods, and thesaurus-based methods. The following is a description of these methods.

3.1.1 Latent semantic analysis

Latent semantic analysis (LSA) is an unsupervised statistical algorithm that is used for extracting and representing relations between words and documents in large unlabeled text corpora [29]. LSA is based on the assumption that there is some underlying (*latent*) semantic structure that is partially concealed by the variability of the contextual usage of words in a certain collection [29]. Formally, LSA starts by constructing a $m \times n$ word–document matrix (χ) for words and documents in the corpus. This matrix is usually huge and sparse. Rows represent unique words in the corpus, and columns represent textual artifacts (full documents or pieces of text). A specific weight is then assigned to each word in the matrix. Such weights can range from simple word counts in documents, to more sophisticated schemes such as the TF.IDF weights of words [76].

Singular value decomposition (SVM) is applied to decompose the *word* $\times$ *document* matrix χ into a product of three matrices $\chi = USV^T$ [31]. $\boldsymbol{S}$ represents a diagonal matrix of singular values (SVD), where these values appear in a descending order, and both $\boldsymbol{U}$ and $\boldsymbol{V}$ are column orthogonal matrices. Dimensionality reduction is then performed to produce reduced approximations of USV^T by keeping the top k eigenvalues of these matrices. A dimensionality reduction technique takes a set of objects that exist in a high-dimensional space and represents them using lower dimensions. This technique is applied to S by removing singular values that are too small (often determined empirically) and only keeping the initial principal components. The main assumption behind this step is that reducing the dimensionality of the observed data will smooth out the data, thus resulting in relatively better approximations to human cognitive relations [50]. The reduced matrix can be described as $\chi_k = U_k S_k V_k^T$, where χ_k represents a compressed matrix of rank k that minimizes the sum of the squares of the approximation errors of χ. Since $U_k S_k V_k^T$ are $m \times k$, $k \times k$, and $k \times n$, respectively, χ_k is still a $m \times n$ matrix similar to χ. Using χ_k, the similarity between two words in the corpus can be measured as the cosine of the angle between their corresponding compressed row vectors.

3.1.2 Thesaurus-based methods

Methods under this category exploit lexical knowledge structures in linguistic databases to discover similarity relations between words. WordNet is the most popular English dictionary that is often used to carry out such analysis. Introduced and maintained by the Cognitive Science Laboratory of Princeton University, WordNet is a large lexical database of English verbs, nouns, and adjectives grouped into sets of cognitive synonyms called synsets [32]. English nouns and verbs in WordNet are connected through a hierarchical structure that models semantic relations such as *hypernym* and *hyponym* among them. Moving up and down the hierarchy reflects how different abstract and concrete concepts are related. This semantic arrangement of concepts gives WordNet an advantage over classical alphabetical dictionaries when it comes to semantic analysis.

In general, similarity methods that exploit the lexical database WordNet can be categorized into three main categories [81], including:

- Path based: These methods estimate the semantic similarity between words based on the shortest path connecting them in the WordNet hierarchy. The shorter the path, the smaller the semantic distance between words. An edge in the word hierarchy represents a uniform distance in terms of semantic similarity. Wu-Palmer [86] is an example of methods that follow this approach. Formally, assuming *depth* is the depth of two given concepts relative to the root node in the WordNet taxonomy, and LCS is the depth of the least common subsumer, or the nearest common node for w_1 and w_2, the similarity between w_1 and w_2 can be described as:

$$\mathrm{Wu}(w_1, w_2) = \frac{2 * \mathrm{depth}(LCS)}{\mathrm{depth}(w_1) + \mathrm{depth}(w_2)} \quad (1)$$

- Information content based: Methods under this category estimate similarity between words using notions of information theory. For instance, the Resnik [74] similarity measure calculates the similarity between two concepts using the information they share in common. This amount of information is indicated by the information content of the concepts that subsume them in the WordNet *is-a* hierarchy, or the lowest node in the *is-a* hierarchy which is a *hypernym* of both words. Formally, assuming $P(c)$ is the probability of encountering c in the corpus and IC is the information content of a certain entity c given by $\mathrm{IC}(c) = -\log P(c)$, Resnik [74] similarity measure defines the similarity between w_1 and w_2 as follows:

$$\mathrm{Resnik}(w_1, w_2) = \mathrm{IC}(\mathrm{LCS}(w_1, w_2)) \quad (2)$$

- Gloss based: Methods under this category use the gloss (definition) of words available in WordNet to estimate similarity. In particular, to quantify similarity between two concepts, their glosses are initially enriched with glosses of related words. A similarity score can then be calculated based on the number of overlapping words in these augmented glosses. Lesk algorithm [53] is an example of a gloss-based measure.

3.1.3 Co-occurrence methods

Co-occurrence methods quantify semantic similarity between words based on their distributional cues in the corpus [10, 57]. The main assumption is that important semantic schemes are inherent in the way words appear in a large sample of text documents. In particular, semantically related words tend to occur in similar contexts [10, 18, 50, 57]. Pointwise mutual information (PMI) and Normalized Google Distance (NGD) are examples of computationally simple and effective methods that are used to estimate semantic similarity between words based on their co-occurrence information. These methods can be described as follows:

- Pointwise mutual information (PMI): Introduced by Church and Hanks [18], and later used by Turney [84] to identify synonym pairs based on Web search results, PMI is an information-theoretical measure of information overlap, or statistical dependence, between two words [18]. Formally, PMI between two words w_1 and w_2 can be measured as the probability of them co-occurring versus their probabilities of occurring in a text collection. Assuming the collection contains N artifacts, PMI can be calculated as:

$$\mathrm{PMI} = \log_2\left(\frac{\frac{C(w_1,w_2)}{N}}{\frac{C(w_1)}{N}\frac{C(w_2)}{N}}\right) \quad (3)$$

where $C(w_1, w_2)$ is the number of documents in the collection containing both w_1 and w_2, and $C(w)$ is the number of documents in which w occurs. PMI is symmetrical, which means that the amount of information acquired about w_2 from observing w_1 is equivalent to the amount of information acquired about w_1 when observing w_2. The value of PMI ranges from $-\infty$ to $+\infty$, where $-\infty$ indicates that the two words are not related, or do not co-occur in any text artifact in the collection.

PMI is intuitive, scalable, and computationally efficient [61, 66]. These attributes have made PMI an appealing similarity method to be used to process massive corpora of textual data in tasks such as information retrieval [61], Semantic Web [80, 84], and text mining [44].

- Normalized Google Distance (NGD): This method uses Google page counts to devise a semantic similarity measure between words based on information distance and Kolmogorov complexity [19]. The main assumption is that the statistical co-occurrence of words in the Web reflects their current similarity status in society, and thus, can give an indication of their similarity. Formally, to estimate the semantic distance between two words w_1 and w_2 using NGD, a Google search query Q is requested for w_1, w_2, and w_1 AND w_2. The semantic distance is then measured as:

$$\mathrm{NGD} = \frac{\max\{\log(D_1), \log(D_2)\} - \log(|D_1 \cap D_2|)}{\log(|D|) - \min\{\log(D_1), \log(D_2)\}} \quad (4)$$

where D_1 and D_2 are the hit counts (number of returned links) of $Q(w_1)$ and $Q(w_2)$, or the number of documents containing w_1 and w_2, respectively, and $|D_1 \cap D_2|$ is the hit count of $Q(w_1$ AND $w_2)$, or the number of Web documents that contain both w_1 and w_2. The main tenet is that pages that contain both words indicate similarity, while pages that contain only one of the words suggest the opposite. NGD is a distance measure, meaning that $\mathrm{NGD}(w_1, w_1) = 0$, and the NGD of two words that do not occur together on any Web page, but do occur separately (i.e., $D_{12} = 0$) is ∞. In order to bound NGD value between 0 and 1, the following formula is often used [38]:

$$n\mathrm{NGD} = e^{-2 \cdot \mathrm{NGD}(w_1, w_2)} \quad (5)$$

NGD has gained momentum in recent years due to its solid theoretical foundation, simplicity, low computational complexity across several applications, and

ability to achieve a decent correlation with the human perception of similarity [15, 85, 87].

3.2 Requirement word clustering

Several word clustering algorithms have been proposed in the literature [7, 79]. These algorithms apply various strategies to group words into semantically coherent clusters. To answer **RQ2**, in our analysis we investigate two families of clustering techniques, including hierarchical and partitioning clustering. Next is a description of these techniques in greater detail.

3.2.1 Partitioning clustering

Partitioning clustering relocates data instances by moving them from one cluster to another, starting from an initial partitioning until a certain error criterion is minimized. k-means is the most popular example of partitioning-based methods. This unsupervised clustering technique seeks to minimize the average distance between data points in the same cluster. Initially, K centroids are randomly selected from the data space. These centroids represent the seeds of the K clusters to be generated. Each point in the space is then assigned to the cluster with the nearest centroid. After all the points have been assigned, the centroids are recalculated as the average of all points in the cluster. The data points in the space are then reassigned to their nearest centroid. This process is repeated until the centroids do not move any more. In other words, the within-cluster distance is minimized.

Identifying the optimal number of clusters and the initial combination of centroids that minimize the within-cluster distance in a certain data space is an NP-hard problem. However, approximate solutions can be found by minimizing the squared error objective function. Formally, given a K number of clusters and a set of n data points, the main objective is to choose the K centroids in such a way that minimizes the total squared distance between each point and its closest centroid, given by:

$$\text{Error} = \sum_{i=1}^{K} \sum_{j=1}^{|K_i|} |x_j - c_i|^2 \tag{6}$$

K is the number of clusters, $|K_i|$ is the number of data points in the cluster K_i, x_j is a data point in the cluster K_i, and c_i is the centroid of that cluster.

In our analysis, words are clustered based on their pairwise similarity that is given in the form of a distance matrix. Therefore, determining new clusters' centroids by averaging data points in the cluster does not make sense. To cluster such data, another derivation of k-means, known as k-medoids, is used. Unlike k-means, k-medoids chooses data points as centroids (medoids). A medoid is the data point for which the average distance to other data points in the set is minimal (i.e., the most centrally located point in the set). Partitioning Around Medoids (PAM) is a classical algorithm that is used to solve the k-medoids problem [47]. This algorithm can be described as a three-step procedure, including:

- Initialize: K points are randomly selected as initial medoids (centroids)
- Build: Each data point x in the space is assigned to the nearest centroid c, creating K clusters. The total cost of the configuration is then calculated as:

$$\text{cost} = \sum_{i=1}^{K} \sum_{j=1}^{|K_i|} |x_j - c_i| \tag{7}$$

- Swap: For each cluster K_i, the centroid c_i is exchanged with a point x_j in K_i and the total cost of the configuration is recalculated. If the cost has increased, undo the swap. Otherwise, if the cost has decreased, the space is rebuilt again around the new centroids (go to step 2). This process is repeated iteratively until no further changes in the centroids can reduce the overall cost.

k-medoids is computationally more expensive than k-means since finding the medoid is usually harder than simply calculating the average. However, k-medoids is known to be more robust to noise and outliers in comparison with k-means as it minimizes a sum of general pairwise distances (dissimilarities) instead of a sum of squared distances. To illustrate k-medoids operation, consider the data space represented in the distance matrix in Table 1. Assuming $K = 2$, the initial medoids are selected to be the words `font` and `color`. Based on their pairwise distance, words `logon` and `password` will be assigned to the medoid word `color`, and `email` will be assigned to the medoid `font`. The overall cost of this configuration (Fig. 2a) is $[(0.93 + 0.86) + 0.8 = 2.59]$. In the second iteration, assuming the medoid `color` is replaced by `logon`, this results in a cost reduction of $[(0.2 + 0.93) +$

Table 1 A semantic distance matrix of the word sample: `email`, `color`, `password`, `font`, and `logon`

	email	color	password	font	logon
email	0.0	0.88	0.12	0.8	0.3
color	0.88	0.0	0.86	0.37	0.93
password	0.12	0.86	0.0	0.91	0.2
font	0.8	0.37	0.91	0.0	0.96
logon	0.3	0.93	0.2	0.96	0.0

0.8 = 1.93]. However, swapping the medoid `color` with `password` will reduce the cost to ((0.2 + 0.86) + 0.8 = 1.86). The latter swap minimizes the overall cost. The space is therefore rebuilt around the new medoids (Fig. 2b). Now `color` is assigned to the medoid `font`, and `email` is reassigned to the medoid `password`. The total cost of the new configuration is [(0.2 + 0.12) + 0.37 = 0.69]. No other possible changes can reduce the cost any further.

3.2.2 Hierarchical clustering

The underlying logic of hierarchical clustering is to successively merge the most similar elements in a data space into larger clusters. Hierarchical clustering algorithms produce hierarchical structures that can be represented as dendrograms—a tree-structured graph that reflects how objects in a space are being grouped at different iterations of the algorithm [3].

Hierarchical agglomerative clustering (HAC) is the most popular family of hierarchical clustering. In our analysis, we experiment with three categories of HAC algorithms, including 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: $M(A,B) = \text{average}\{d(a,b) : a \in A, b \in B\}$

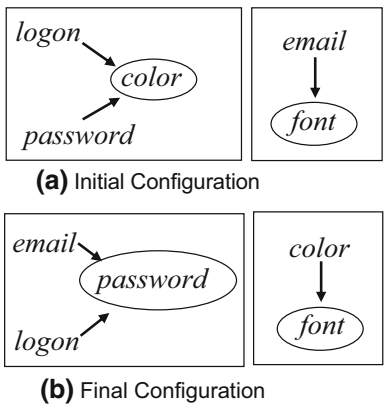

Fig. 2 k-medoids clustering of the data in Table 1. **a** Initial configuration (cost = 2.59), **b** final configuration (cost = 0.69)

To demonstrate the HAC process, we refer to the data space in Table 1. Figure 3 shows the generated dendrogram using average linkage algorithm. First, each word is assigned to a separate cluster. The two closest words in the space `email` and `password` are merged. At the second step, the word `logon` is merged with the cluster <`email`, `password`> since its average distance to this cluster is ((0.3 + 0.2) / 2 = 0.25), which is smaller than the distance to the two other clusters <`color`> and <`font`>. At the third step, words `font` and `color` are merged as they are closer to each other than to the cluster <`email`, `password`, `logon`>. Eventually, all clusters are merged into one big cluster.

4 Implementation, analysis, and evaluation

In this section, we analyze the performance of the different semantic similarity methods and clustering techniques introduced in Sect. 3 to approximate proper cluster configurations for our approach (i.e., answering research questions **RQ1** and **RQ2**). In particular, we describe our experimental systems and our evaluation strategy, provide technical details about the implementation of the different methods, and discuss the analysis results.

4.1 Experimental systems

Three experimental software systems from different application domains are used to conduct our analysis. These Java 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 our confidentiality agreement, 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 USA. The application exploits several third-party APIs to make routing and accommodation recommendations.
- *SafeDrink*: A software system that is designed to help users to manage their drinking habits. The system has a

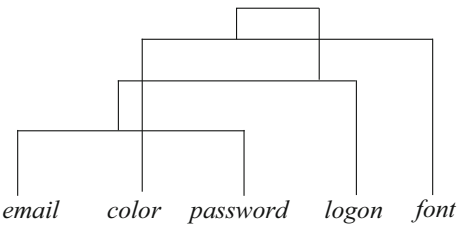

Fig. 3 Hierarchical clustering of the data in Table 1

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 which provides users with options to plan their budgets and manage their personal finances, including their credit cards, banking accounts, and tax information.

The functional requirements of each system were extracted from the SRS document maintained for each project. In particular, individual features (sections in the SRS document) of each system were first identified. Diagrams, indexes, and references were removed. Each atomic specification of each feature was then treated as a separate functional requirement. Each requirement is around one or two sentences and often starts with phrases such as *The system shall* or *The user must*. For instance, the generate map feature of the *SmartTrip* system was broken down into a set of FRs including:

```
The system shall establish a connection ..
The system shall display most recent ..
The user can zoom into different  ..
Zooming should be enabled through ..
```

In total, we were able to extract 170, 214, and 184 unique FRs from *SmartTrip*, *SafeDrink*, and *BlueWallet*, respectively. Table 2 summarizes 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 (CLOC), number of functional requirements (FR), number of unique words in each system's requirements (FR. WORDS), number of code classes (CLS), and number of developers available to participate in the study (DEVL).

4.2 Measuring cluster quality

Recent analysis in NLP research shows that groups of words that have a higher average of pairwise semantic similarity between their words tend to be more semantically coherent, thus more meaningful to end users [62, 65]. Similarly, words that share a higher average of semantic similarity with other words in the cluster tend to be more representative of the cluster's subject matter [49, 51]. Based on these observations, in our analysis, the semantic coherence of a cluster c_i of size n words (w_i) is calculated as the average pairwise semantic similarity between its words:

$$\text{cohesion}(c_i) = \frac{1}{\binom{n}{2}} \sum_{w_i, w_j \in c, i \neq j} \text{similarity}(w_i, w_j) \tag{8}$$

Furthermore, meaningless clusters are penalized. In particular, any generated cluster with less than 8 words is assigned a cohesion of 0. This heuristic is enforced based on previous observations that clusters with less than 8 words hardly convey any meaningful concepts [14, 62, 65]. Therefore Eq.(8) can be expanded as follows:

$$\begin{cases} 0 & n < 8 \\ \dfrac{1}{\binom{n}{2}} \displaystyle\sum_{w_i, w_j \in c, , i \neq j} \text{similarity}(w_i, w_j) & n > = 8 \end{cases} \tag{9}$$

The separation, or coupling, of a cluster is quantified as the average pairwise word semantic similarity between a cluster and its nearest neighbor in the generated cluster space. Formally, assuming M number of clusters, coupling of c_i is calculated as follows:

$$\text{coupling}(c_i) = \max_{0 \leq j \leq M, j \neq i} \text{SIMILARITY}(c_i, c_j), \tag{10}$$

where $\text{SIMILARITY}(c_i, c_j)$ is the average word pairwise similarity between clusters c_i and c_j. The overall quality of the generated cluster space is then calculated as:

$$Q(C) = \frac{1}{|C|} \sum_{i=1}^{|C|} (\text{cohesion}(c_i) - \text{coupling}(c_i)) \tag{11}$$

The value of this semantically aware quality metric fits in the range $[-1, 1]$. 1 indicates a perfect scenario, where cohesion is maximized and coupling is minimized and -1 is a worst-case scenario, where formed clusters have no semantic coherence. Given this objective quality function, the main task is to identify cluster configurations that can actually generate a solution (converge) given the different semantic distance measures introduced earlier.

Table 2 Experimental systems

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

4.3 Identifying cluster configurations

To evaluate the performance of the different semantic similarity methods, similarity matrices are generated for the words extracted from the FRs of each experimental system using each of the methods discussed earlier (Resnik, Wu, Lesk, LSA, PMI, and NGD).

Table 3 An example of text lemmatization, stemming, and stop-word removal

Original sentence	Only adjusters with a supervisor role can update preferred repair facility ratings
OpenNLP lemmatizer	Only adjuster with a supervisor role can update prefer repair facility rating
Porter stemmer	Only adjust with a supervisor role can update prefer repair facility rate
Stop-word removal	Only adjuster supervisor role update preferred repair facility rating

To implement LSA (i.e., perform SVD), the Bluebit Matrix Calculator[1], a high-performance matrix algebra for .NET programming, is used. To build the LSA *word* × *document* matrix of the system, each FR is treated as a separate document. FRs are initially stemmed using Porter stemmer [73]. English stop-words are then removed, and words and documents are arranged in a $m \times n$ matrix, where each row represents a unique FR word (m_i), and each column (n_j) represents a system document (an individual FR in our analysis). The matrix entry *i.j* is the frequency of m_i in n_j.

The Java API *WordNet::Similarity*[2] is used to implement the list of thesaurus-based semantic similarity methods (i.e., Wu, Lesk, and Resnik). Initially, each FR of the system is extracted and lemmatized. The goal of lemmatization is to reduce the inflectional forms of a word to a common base, or a linguistically valid form, known as the lemma. Stanford OpenNLP lemmatizer is used in our analysis. The reason lemmatization is selected over-stemming, for this particular set of methods is to preserve the naturalness of words. In particular, popular stemmers such as Porter are prone to over-stemming [73] which happens when 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*). This can be a key factor in the performance of methods that rely on English dictionaries for similarity calculations. For example, Table 3 shows the resulting text after applying lemmatization and stemming over a sample FR. When using Porter stemmer, certain words such as `updat`, `repair`, and `facil` are over-stemmed to unnatural words. However, using lemmatization, all generated words are still valid dictionary words. After lemmatization, English stop-words are removed (e.g., *the*, *was*). The pairwise similarity between remaining words is then calculated using the different similarity methods provided in the *WordNet::Similarity* API.

To implement the co-occurrence method PMI, the *word* × *word* matrix of the system is constructed based on the words' co-occurrence patterns in the system's FRs. This matrix size is $n \times n$, where n is the number of unique words in the system. Note that since co-occurrence is a symmetrical relation, the upper and the lower triangles of the PMI/NGD_{Wiki} co-occurrence matrices are identical, thus reducing the space requirements by half.

The Google API[3] is often used to build the hit-count matrix for NGD [33]. This API provides developers with an interface to integrate Google services in their programs. However, Google 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 methods that rely solely 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 unrepeatable manner, producing different counts at different time periods or different search engines [83]. Moreover, due to the high ratio of noisy content (e.g., random ad placement) on the Web, some words might occur together arbitrarily, or due to pure error [8], thus raising major concerns about the reliability of such counts for similarity estimations [33]. In order to control for such effects, we constrain the co-occurrence data search process to Wikipedia (i.e., NGD_{Wiki}). Even though Wikipedia is naturally a smaller corpus of human knowledge than the entire World Wide Web, it is still the largest freely available, semi-structured, and constantly evolving source of human knowledge. Moreover, Wikipedia is often updated by experts, which makes it less noisy in comparison with the entire Web [34, 82]. To implement NGD using Wikipedia, we downloaded the most recent XML dumps of the English Wikipedia[4]. Individual articles were then extracted and indexed by lemmatization and removing English stop-words and hyperlinks. Indexed artifacts were then stored in a local database. The total number of extracted articles was 4.5 millions, and the database size on disk was 26.7 GB.

The generated pairwise semantic similarity matrices of each of our experimental systems are fed to the different clustering algorithms discussed earlier (k-medoids, AL, CL, and SL). The performance of these different cluster configurations is then assessed using the quality measure defined in Eq. (11). To approximate a near-optimal number of clusters (K) in k-medoids, K is initially set to 2. The system's FR words are then clustered, and the quality function Q is measured. K value is then increased by 1, and the process is repeated. The value that generates the best performance is considered. In HAC algorithms, the number of clusters is initially set to n, which is the number of words

[1] http://www.bluebit.gr/net/.

[2] http://wn-similarity.sourceforge.net/.

[3] https://developers.google.com/web-search/docs/.

[4] http://en.wikipedia.org/wiki/Wikipedia:Database_download.

Table 4 Results of the optimization process using k-medoids clustering algorithm

System	Resnik		Wu		Lesk		LSA			PMI		NGD_{Wiki}	
	Q	K	Q	K	Q	K	Q	K	LSA-*k*	Q	K	Q	K
SmartTrip	0.09	21	0.08	13	0.09	12	0.07	32	60	0.01	15	0.03	22
SafeDrink	0.09	15	−0.25	22	−0.12	33	0.08	6	110	−0.04	24	−0.02	11
BlueWallet	−0.3	7	0.01	14	0.07	43	−0.1	12	130	−0.03	31	−0.1	7

Table 5 Results of the optimization process using complete linkage clustering algorithm

System	Resnik		Wu		Lesk		LSA			PMI		NGD_{Wiki}	
	Q	K	Q	K	Q	K	Q	K	LSA-*k*	Q	K	Q	K
SmartTrip	0.1	17	0.01	16	0.1	6	0.1	28	60	0.09	21	0.2	22
SafeDrink	0.07	19	0.05	24	0.03	14	0.11	23	100	0.08	41	0.11	18
BlueWallet	0.07	24	0.08	31	0.04	26	0.1	14	80	−0.08	15	0.14	33

Table 6 Results of the optimization process using single linkage clustering algorithm

System	Resnik		Wu		Lesk		LSA			PMI		NGD_{Wiki}	
	Q	K	Q	K	Q	K	Q	K	k	Q	K	Q	K
SmartTrip	0.04	13	−0.11	68	−0.13	97	0.14	17	60	−0.12	105	0.19	21
SafeDrink	0.01	44	0.04	48	0.07	43	0.17	43	110	−0.03	65	0.22	22
BlueWallet	0.05	117	−0.05	135	−0.06	74	0.24	37	160	−0.17	74	0.3	44

Table 7 Results of the optimization process using average linkage clustering algorithm

System	Resnik		Wu		Lesk		LSA			PMI		NGD_{Wiki}	
	Q	K	Q	K	Q	K	Q	K	k	Q	K	Q	K
SmartTrip	0.04	12	−0.07	14	−0.03	40	0.41	22	80	−0.1	103	0.67	16
SafeDrink	−0.09	6	0.04	3	−0.2	110	0.47	18	90	0	10	0.63	26
BlueWallet	−0.09	33	0.04	22	−0.03	130	0.32	42	130	−0.13	220	0.57	33

extracted from the system's FRs. To determine the optimal cut point in the generated dendrogram at which to stop the merging process, the quality function is measured at each level. The level (number of clusters) that maximizes the quality function is considered as the optimal cut point.

It is important to point out that optimizing clustering results using LSA is more complicated than other similarity measures. In particular, the LSA$-k$ value has to be initially determined before the LSA pairwise similarity distance matrix of the system is generated. In our analysis, we adopt a brute-force strategy to approximate such values. Initially, LSA$-k$ is set to 10, the LSA pairwise similarity matrix is generated, and the optimization process is executed. The value of LSA$-k$ is then gradually increased by 10, and the process is repeated until LSA$-k$ is equal to the number of singular values in **S** of the system (Sect. 3.1.1). The step size of 10 is determined as the minimum step that has a noticeable impact on the quality.

Given the complexity of the optimization process (3 systems $\times$ 4 clustering algorithms $\times$ 6 similarity measures), the entire evaluation process is automated. It is important to point out that a brute-force solution is only feasible in our experiment because the number of data items (words) in the space is relatively low (Table 2). However, this should not affect the scalability of our approach given that requirements are often described using limited domain terminology [36, 60].

The results of the optimization process over our three experimental systems are shown in Tables 4, 5, 6, 7. Due to space limitations, we only report the configurations that maximized the value of our quality function (Eq. 11). Next we discuss these results in greater detail.

4.4 Results and analysis

To get a sense of the different similarity methods' and clustering algorithms' performance, we average the generated results, in terms of cluster quality (Eq.11), over all our experimental systems. The results, displayed using boxplot charts in Figs. 4 and 5, show the aggregated average performance of the different clustering algorithms and similarity methods, respectively (i.e, **RQ1** and **RQ2**).

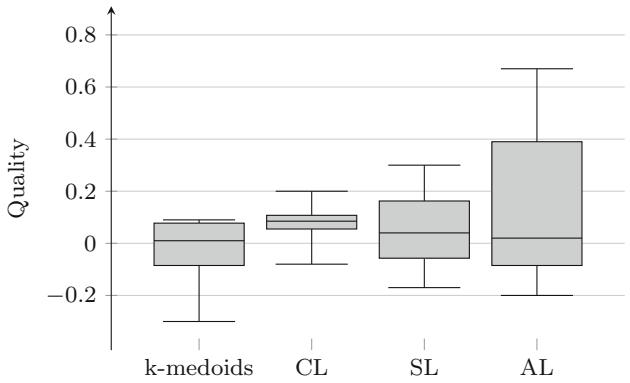

Fig. 4 Aggregated performance of the clustering algorithms (k-medoids, CL, SL, and AL) in terms of quality (Eq. 11)

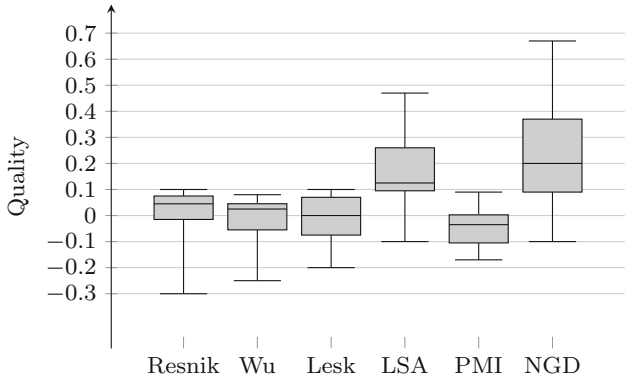

Fig. 5 Aggregated performance of the similarity methods (Resnik, Wu, Lesk, LSA, PMI, and NGD_{Wiki}) in terms of quality (Eq. 11)

The aggregated clustering results in Fig. 4 show that k-medoids was almost random in its operation, failing to converge, or achieve competitive results in all three systems. The family of HAC algorithms was relatively more successful in finding cohesive clusters. In particular, the results show that average linkage (AL), using NGD_{Wiki} and LSA as the underlying similarity measures, was the most successful in extracting highly cohesive word clusters in all systems. CL and SL, however, could not match AL's performance, 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 [2]. CL and SL, on the other hand, rely on a single word pair to determine the distance between two clusters, which seems to affect the word clustering process negatively as other words that might be more distinctive to the cluster may be ignored.

In general, the set of hierarchical clustering algorithms produced higher-quality results in all systems, which answers our first research question (**RQ1**). The family of k-means, represented by k-medoids in our analysis, was not as successful. This can be explained based on the fact that k-means algorithms tend to be extremely sensitive to the initial seeds. For instance, reconsider the semantic distance matrix in Table 1. Assuming the initial seeds for a k-medoids space were selected to be the words `email` and `password`, based on their pairwise distance, the words `color` and `logon` will be assigned to the centroid `password` and the word `font` will be assigned to the centroid `email`. The cost of this configuration is [(0.2 + 0.86) + 0.8 = 1.86]. In the second iteration, assuming the centroid `password` is replaced by the word `logon`, this results in a cost increase of [(0.2 + 0.93) + 0.8 = 1.93]. Similarly, swapping the centroid `color` with the word `password` will increase the cost to [(0.93 + 0.86) + 0.8 = 2.59]. Therefore, the algorithm terminates at this point keeping the original decomposition shown in Fig. 6. This example shows how poor initialization of the clusters will lead to poor clustering results, causing a major drop in the cohesion and coupling of generated decompositions. A suggested solution for this problem would be to run the algorithm as many times as possible and to consider the best performance, or to use seeding techniques to pick potentially good initial seeds [6]. However, adding such parameters to the optimization problem increases the complexity to exponential levels.

In terms of similarity measures (**RQ2**), the aggregated results in Fig. 5 show that LSA and NGD_{Wiki} were in general more successful than the set of thesaurus-based methods and the co-occurrence method PMI. To explain this difference in the performance, we use the word sample in Table 1 (`email`, `logon`, `password`, `color`, and `font`). We calculate the similarity of the word `email` to the rest of the words in the sample. A human with a common knowledge of Web technologies would rank the words `password` and `logon` as more similar to the word `email` than `color` and `font`. Table 8 shows how the different similarity methods ranked these words. Our first observation is that the set of thesaurus-based methods behaved almost the same. In particular, Wu, Lesk (which expresses similarity in integer numbers), and Resnik, ranked the word `logon` at the bottom of the list, with no similarity to the word

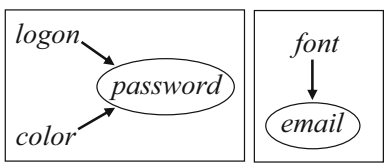

Fig. 6 Example of bad seeding of k-medoids using the data in Table 1

Table 8 Similarity of the words `logon`, `password`, `color`, and `font` to the word `email` according to the different similarity methods

Resnik		Wu		Lesk		LSA		PMI		NGD_{Wiki}	
word	sim	word	sim	word	sim	word	sim	word	sim	word	sim
password	0.78	color	0.38	font	41	password	.52	password	.083	password	0.49
color	0.78	password	0.33	color	35	logon	.14	logon	.083	logon	0.355
font	0.77	font	0.3	password	16	color	.02	color	.083	font	0.23
logon	0	logon	0	logon	0	font	.01	font	.083	color	0.20

`email`. This can be attributed to the fact that this word does not exist in the WordNet dictionary. In fact, the lack of named entities is regarded as one of the most dominant problems of thesaurus-based methods. In particular, due to the explosive growth of the amounts of textual data being shared online, natural language is evolving in a pace like never before. Expert-generated linguistic databases such as classical dictionaries or semantic knowledge bases (e.g., WordNet) cannot keep up with such a fast pace of change; rather, they record snapshots at various stages of that evolution. Therefore, relying on such databases for text analysis might lead to misleading, and often out-of-date, results.

The table also shows that PMI was almost random in its operation. This can be explained based on the fact that PMI struggles with smaller corpora. In particular, the amount of contextual knowledge often available in an SRS document is not sufficient enough for PMI to function probably. For instance, the words `email` and `logon` do not appear together in any FR in the dataset in [24], thus taking PMI accuracy to lower levels, which also explains why additional training data greatly enhances the performance of PMI [9]. In contrast, LSA was able to achieve more sensible results than PMI. However, specifying the LSA-k value that achieved these results was a computationally expensive process. Basically, we had to run LSA under different k values [10–300] to identify the value that achieved the desired results (k = 50). Finally, the table shows that NGD_{Wiki} was able to detect sensible similarity relations that reflect the way these words are typically used by humans. In general, Wikipedia is an up-to-date source of human knowledge and thus does not suffer from coverage issues often associated with linguistic dictionaries. For instance, the word `logon` appears in 299 Wikipedia articles, while the word `email` appears in 16,735 articles. Both words appear together (co-occur) 117 times. Given that our version of Wikipedia has 4.5 million articles, nNGD(`email`, `password`) = 0.36. `Password`, however, appears in 4125 articles and co-occur with `email` in 1337 articles, thus nNGD(`email`, `password`) = 0.49.

5 Detecting NFRs

After FR words in each system have been clustered using the cluster configurations identified in the previous section, the main task at this step of the proposed procedure 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 the generated clusters are classified into various NFR categories. Next is a description of our classification procedure.

5.1 NFR classification

To guide the classification process, a list of classification labels that describes the basic types of NFRs is used. Even though there are almost 150 recognized categories[17], only the most general categories of NFRs that have been identified in the literature are used [24, 78]. These categories include: *security*, *performance*, *accessibility*, *accuracy*, *portability*, *safety*, *legal*, *privacy*, *reliability*, *availability*, and *interoperability*.

To assign word clusters to these NFR 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 9) 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 $\{w_1, w_2, w_3, \ldots, w_N\}$, and a non-functional requirement (L), with M number of morphological variants $\{l_1, l_2, l_3, \ldots, l_M\}$, the probability that C belongs to L is given by:

$$P(C, L) = \frac{1}{N} \sum_{i=1}^{N} \max_{0 \le j \le M} \mathrm{NGD}_{\mathrm{wiki}}(w_i, l_j), \tag{12}$$

After calculating the similarity of a certain word cluster to all NFR labels, the cluster is assigned to the label with the maximum average pairwise similarity. Note that in our

Table 9 NFR categories and their representative words

NFR	Morphological variants
Security	Security, secure
Performance	Performance, perform
Accessibility	Access, accessible, accessibility
Accuracy	Accuracy, accurately, accurate
Portability	Portable, portability
Safety	Safe, safety
Legal	Legally, legal
Privacy	Privacy, private
Reliability	Reliable, reliability
Availability	Availability, available
Interoperability	Interoperable, interoperability

analysis, we adopt crisp clustering, where a word can belong to one cluster only. This constraint is enforced to ensure that the classification process produces more clear-cut results. For instance, certain *popular* English words (e.g., `performance`) tend to have high semantic similarity to multiple NFR labels; therefore, if this word was allowed to belong to multiple clusters, it might overshadow other, less popular, but more deterministic words (e.g., `logon`), thus taking the overall classification accuracy down.

The functional requirements of each system are then automatically classified into different NFR categories based on their semantic similarity to the clusters of words representing individual NFRs. The task at this step is treated as a standard IR problem. In particular, the words of each individual FR are extracted and matched with the text (word cluster) of each detected NFR of the system using the text semantic similarity measure (TSS) proposed by Mihalcea et al. [61]. 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 $\mathrm{TSS}(T_1, T_2)$ can be described as follows:

$$\frac{1}{2}\left(\frac{\sum(\mathrm{maxSim}(w_i, T_2) \times \mathrm{IDF}(w_i))}{\sum \mathrm{IDF}(w_i)} + \frac{\sum(\mathrm{maxSim}(w_j, T_1) \times \mathrm{IDF}(w_j))}{\sum \mathrm{IDF}(w_j)}\right) \tag{13}$$

$\mathrm{maxSim}(w_i, T_2)$ is a function that returns the $\mathrm{NGD}_{\mathrm{Wiki}}$ similarity score between w_i from T_1 and its most similar word in T_2, and $\mathrm{IDF}(w_i)$ is the word's specificity, calculated as the number of documents in the corpus divided by the number of documents that contain w_i.

We notice that, using Eq. (13), an FR might be assigned to an NFR category due to noise. For instance, due to its massive size, some words might co-occur in Wikipedia due to pure error. This might contribute to the similarity of words even if they are not semantically related. Therefore, a similarity threshold has to be enforced to filter out noise. Formally, an FR will be classified under a specific NFR category if and only if their pairwise TSS similarity exceeds a specific similarity threshold.

To determine this threshold, developers from different experimental systems were asked to manually classify the system's FRs into the different NFR categories identified earlier (Table 9), creating our ground-truth dataset. Each functional requirement can be classified under multiple NFRs; therefore, we have a (*M*:*N*) relation between FRs and NFRs. For instance, the *SafeDrink's* FR: *"According to state regulations, text messages exchanged between the user and the designated hcp should be encrypted"* enforces both a *security* and a *legal* constraints. In cases of conflicts (i.e., different developers classify a certain functional requirement under different NFRs) the researchers intervened to help making a decision. To minimize the experimental bias, researchers' involvement was limited to initiating further discussion to help developers reach a consensus. Only a few cases of conflict were reported, and a consensus was reached in all of these cases. The different types of NFRs were explained to the participants, and the task was demonstrated by the researchers using sample functional requirements prior to the experiment. No time constraint was enforced.

To assess the classification accuracy of the proposed approach under different similarity thresholds, recall and precision values are calculated after all the FRs in the system are classified. Recall of each NFR 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 [24, 78]. Precision and recall are measured at different similarity thresholds (λ). Figure 7 shows the classification results in all three systems over the λ values of $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. The results are reported using the F measure, which represents the harmonic mean of recall and precision. Formally, the F measure is described as follows:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{14}$$

In our analysis, we use $\beta = 2$. F_2 emphasizes recall over precision. The main assumption is that, from a practicality point of view, commission errors (false positives) tend to be easier to deal with than omission errors (false negatives) [24].

Figure 7 shows the classification results in terms of F_2 averaged over all the NFRs in all of our experimental

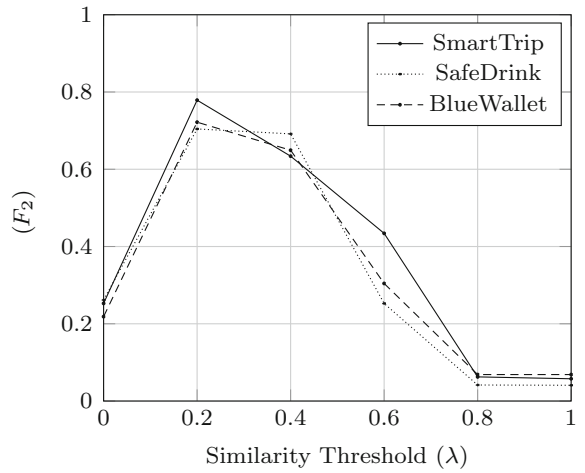

Fig. 7 The performance of our classification procedure in terms of F_2 at different similarity thresholds (λ)

systems. These results include also FRs which do not enforce any NFR (*other*). An FR that does not enforce any NFR and is classified under *other* is considered a correct case of classification. At $\lambda = 0$, all FRs in all systems are classified under all NFRs. In this case, recall is maximized except for the *other* case; however, precision is very low. In contrast, at $\lambda = 1$, none of the FRs are classified (all FRs are assigned to *other* category). In this case, both recall and precision are 0 except for the *other* case.

In general, results show that the best performance in all systems was detected in the λ range of [0.2–0.4]. At $\lambda > 0.4$, less and less FRs are being assigned to NFR categories; thus, while the precision might be increasing, the recall is deteriorating, which explains the very low F_2 values. In contrast, at $\lambda < 0.2$, almost every FR is getting assigned to one or more FRs, which explains the high recall, but the very low precision. Since we favor recall over precision, the performance seems to be better (in terms of F_2) at this level than higher threshold levels. However, it is practically useless as the average precision is very low.

5.2 Analysis

Table 10 shows the classification performance, in terms of precision and recall, achieved in our three experimental systems at a threshold similarity $\lambda = 0.3$. The results show that our procedure managed to achieve an average classification recall of 88, 74, and 57 % and an average classification precision of 52, 57, and 50 % in *SmartTrip*, *SafeDrink*, and *BlueWallet*, respectively. These results can be interpreted by analyzing individual cases. For instance, *SafeDrink* enforces a high *security* constraint. This NFR is characterized by the word cluster {`password`, `login`, `email`, `authenticate`, `authorize`, `log`, `register`, `protect`, `access`, `allow`, `role`}. These words appear in functional requirements such as *"registered users should login using their email and password"* and *"the system shall authenticate users' access credentials."*

In the *BlueWallet* system, the *security* constraint is emphasized through the encryption algorithm the system uses to protect its users' data. In particular, this system deals with sensitive user information. It requires encryption of all the 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`, `protect`}. 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 *"the total should be rounded to two decimal places."*

SafeDrink maintains a list of different states' drinking laws and provides information about local law enforcement agencies and current drinking regulations in the area, thus enforcing a *legal* constraint. This particular NFR is captured by the word cluster {`law`, `lawyer`, `regulation`, `guidelines`, `legal`, `police`, `insurance`, `standard`, `comply`, `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 its 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`, `red`, `blue`, `green`, `cup`, `access`, `look`, `feel`, `simple`, `appealing`}. *SmartTrip*, however, 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`, `second`, `hour`, `trans`, `transmit`, `signal`, `live`}.

In terms of limitations, *SmartTrip* and *SafeDrink* enforce an *interoperability* constraint. This NFR is enforced by functional requirements that manage the interaction between the client side interface, the server, and the third-party APIs. We notice that the clustering algorithm failed to group keywords with semantic similarity to the word *interoperability*. This can be attributed to two different

Table 10 NFR classification performance at $\lambda = 0.3$

NFR	SmartTrip					SafeDrink					BlueWallet				
	FR	TP	FP	R	P	FR	TP	FP	R	P	FR	TP	FP	R	P
security	2	2	2	1	0.5	23	23	11	1	0.68	24	23	24	.96	0.52
performance	33	33	20	1	0.62	9	7	3	0.78	0.7	12	11	12	0.92	0.48
accessibility	4	4	8	1	0.33	32	15	11	0.47	0.58	8	8	14	1	0.36
accuracy	9	7	12	0.78	0.37	11	8	4	0.73	0.67	36	36	14	1	0.72
portability	16	16	21	1	0.43	21	14	4	0.67	0.78	2	2	4	71	0.33
safety	10	10	13	1	0.43	21	14	11	0.67	0.56	NA	NA	NA	NA	NA
legal	6	6	12	1	0.33	38	28	7	0.74	0.8	16	14	13	0.88	0.52
privacy	9	8	14	0.89	0.36	12	9	9	0.75	0.5	9	8	9	0.89	0.47
reliability	21	20	9	0.95	0.69	4	2	22	0.5	0.08	14	12	11	0.86	0.52
availability	7	4	2	0.57	0.67	NA	NA	NA	NA	NA	4	2	4	0.5	0.33
interoperability	19	18	13	0.95	0.58	14	12	14	0.86	0.46	NA	NA	NA	NA	NA
other	53	21	1	0.4	0.95	29	27	22	0.93	0.55	69	33	10	0.48	0.77
average				0.88	0.52				0.74	0.57				0.86	0.50

FR is the number of functional requirements classified manually under the associated NFR category, TP is the number of true positive classifications and FP is the number of false positives, and P is precision and R is recall

reasons. First, the word *Interoperability* appears in a relatively smaller number of Wikipedia articles (63 articles). Therefore, its chances to co-occur with other words are limited. Second, words that are often associated with this particular NFR tend to be scattered over the clustering space and are often grouped with clusters related to *performance* and *portability*. Another example of potential failure is polysemy, or when a 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 $NGD_{Wiki}(accuracy, fault) = 0.25$ and $NGD_{Wiki}(reliability, fault) = 0.24$. This problem can be probably controlled by considering phrases rather than single words. However, such claims are yet to be evaluated in our future work.

6 NFR traceability

In the second phase of our analysis, we trace discovered NFRs to their implementation in the solution space. The main objective is to generate NFR traceability information with accuracy levels that can be adequate for practical applications. NFR-to-code traceability helps to verify that all quality constraints in the system have been implemented, aid in tasks such as regression testing, verification and validation, and change impact analysis [23, 37, 63].

In our analysis, we adopt an IR approach for tracing NFRs. Our approach is enabled by the fact that, in the first phase of our analysis, NFRs were extracted and explicitly defined as clusters of FR words. Therefore, they can be matched with other artifacts in the system based on their textual content. The underlying tenet is that developers use common vocabulary to describe domain concepts; thus, software artifacts sharing similar textual information are likely to be traceable from one another [27, 45].

6.1 IR-based traceability

The main research question at this phase of our analysis is how to trace the discovered NFRs to the source code artifacts implementing them (**RQ3**). To answer this question, we investigate the performance of three different IR methods in establishing and recovering similarity relations between source code artifacts and NFR clusters. These methods include:

- Text semantic similarity (TSS): Based on our findings in the first phase of our analysis, we adopt TSS (Eq. 13) [61] as an IR method to match discovered NFRs with code entities. TSS was originally introduced to measure semantic similarity between short texts. Short text is a relatively recent NLP classification of text that has been motivated by the explosive growth of micro blogs on social media (Twitter, YouTube comments, status updates, instant messaging), and the urgent need for effective methods to analyze such large amounts of limited textual data [40]. The main characteristics of

short text include data sparsity, noisy content, and colloquial terminologies. Such characteristics can also be observed in source code, including the limited vocabulary that is used in the code, the noisy content generated by arbitrary naming conventions, and the colloquial terminologies emerging from domain-specific acronyms and abbreviations [43, 72]. Therefore, we find it appropriate here to use this method to match NFRs with code.

- Vector space model (VSM): VSM is an algebraic model that describes and compares objects using N-dimensional vectors, where each dimension corresponds to an orthogonal feature of the object [76]. Formally, using VSM, each document in the collection is represented as a set of weighted terms $T = \{t_1, \ldots, t_n\}$. The terms in T are regarded as the coordinate axes in an N-dimensional coordinate system, and the term weights are their corresponding values. If q and d are two artifacts represented in the vector space, then their similarity can be measured using the cosine of the angle between their vectors:

$$\mathrm{Sim}(q,d) = \frac{\sum q_i \cdot d_i}{\sqrt{\sum q_i^2 \cdot \sum d_i^2}} \quad (15)$$

  where q_i and d_i are real numbers representing the TF.IDF weights (importance) of the term i in q and d, respectively [76]. VSM with TF.IDF weights is often used in traceability studies as an experimental baseline [45, 60].
- Latent semantic indexing (LSI): LSI is a derivation of LSA that is used for calculating documents' similarity. The underlying assumptions are similar to LSA. However, LSI considers the document vectors in the LSA semantic space (Sect. 3.1.1). More specifically, a document is represented as a vector of weighted corpus words. The similarity between two documents can then be calculated as the cosine similarity between their vectors in the LSI-SVD reduced space [29, 75]. Using LSI, two documents can have a high similarity even if they do not have any overlapping words, as long as their words are semantically similar in the latent semantic space.

In our analysis we work at class-granularity level, where individual NFRs are matched with code classes potentially implementing them [22]. This decision is supported by the observation that higher granularity levels (e.g., a code snippet or a method) do not provide sufficient information for a method such as TSS or LSI to work [10]. 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 IR methods of context. In what follows, we describe in detail the evaluation process of these different methods.

6.2 Pre-processing and evaluation

NFRs are represented in our analysis as clusters of natural language words. To ensure an accurate retrieval process, the natural language content of source code, embedded in identifiers, comments, and string literals, must be initially extracted. In software systems, comments and code messages are mainly used to communicate the code, its features (e.g., help messages), and errors (e.g., exception messages) with other programmers and end users of the system. Therefore, they are often expressed in natural language and can be simply indexed by lemmatization and stop-word removal. However, this process is more challenging when dealing with code identifiers (e.g., method, variable, and class names). Identifiers appearing in a software system can be natural language words, unnatural tokens (e.g., abbreviations, acronyms, arbitrary names), or a combination of both. Camel-casing is often used to split identifiers into their constituent words (e.g., `UserID` is split to `User` and `ID`). However, recent studies of code indexing have revealed that source code often contains a considerable percentage of identifiers with names that are not properly camel-cased. Therefore, relying solely on camel-case splitting might lead to information loss [42].

To overcome this problem, we propose an algorithm that looks for natural language words in code identifiers. The algorithm recognizes two types of tokens, including natural language words (dictionary words) and unidentified tokens (abbreviations, acronyms, or arbitrary alphanumerical sequences). The algorithm attempts to identify the split that generates the least number of unidentified tokens. Formally, at the first iteration, the algorithm scans the input string (identifier name) from left to right starting from the first character. After each character scanned, a candidate split is taken. The algorithm checks whether the left or the right parts of the string are a natural word. In particular, the algorithm first looks for the word in the list of the system's FR words. If it is not found, the algorithm uses the Ispell English word list[5] to identify natural words. If both parts of the string after the split are words, the algorithm terminates. If the split did not result in any natural words, one more character is scanned. If either the left or the right parts of the string are a natural word, that word is returned and the algorithm is re-executed on the other part of the string that is not a natural word. If the entire input string is scanned and no words are identified, the algorithm executes another iteration, scanning the input string starting from its second character, and so on.

[5] http://wordlist.aspell.net/.

To demonstrate this algorithm's operation, consider the variables `requesthcprrecord`, `setThemecolor`, and `cptvehicletypestr`. All these identifiers are defined with no proper camel-casing. The algorithm's operation over the identifier `requesthcprrecord` is shown in Table 11. The input string is read from left to right starting from the first character. The first valid split identified by the algorithm is {`request hcprecord`}. The algorithm re-executes over the right part of the string `hcprecord`, and the split {`hcp record`} is identified. The algorithm terminates at this point as the renaming part of the string (`hcp`) is less than 4 characters long, and it is highly unlikely that it contains a meaningful natural word.

For the identifier `setThemecolor`, the algorithm initially splits this string based on camel-casing into `set` and `Themecolor`. The algorithm then scans the substring `Themecolor` from left to right looking for natural words. The first word `The` is detected after the split {`The mecolor`}. This splits generates one natural language word and one unidentified token. The algorithm continues until the split {`Theme color`} is reached. This split divides the input string into two natural words and no unidentified tokens. The algorithm terminates at this point. Similarly, for `cptvehicletypestr`, the first three iterations of the algorithm do not detect any natural words. The fourth iteration, starting from character ''`v`,'' detects the string `vehicle` in the split {`cpt vehicle typestr`}. The algorithm then re-executes over `typestrg`. The split {`type str`} is taken. The algorithm then terminates as the remaining substrings are less than 4 characters long. In cases where there are many possible valid splits, the algorithm chooses the split that generates the least number of natural language words. For example, consider the case `officevisit` the algorithm finds two valid splits `off ice visit` and `office visit`. Both of these splits are equally valid since they only generate natural language words. However, the algorithm picks the second split as it returns longer natural language words. This heuristic has been found to generate more accurate results.

The outcome of the indexing process is compact content descriptors, or *profiles*, containing keywords of individual code classes. To answer **RQ3**, for each experimental system, the discovered NFR clusters are matched with all the code class profiles in the system using TSS, VSM, and LSI. Retrieved classes are then ranked in a list of candidate traceability links based on their similarity to the NFR being traced. For LSI, the *word* × *document* matrix is constructed considering each code class profile and each NFR word cluster as separate documents. To run TSS, natural language words from the system's class profiles are tabulated in the system's word co-occurrence matrix based on their Wikipedia occurrence patterns with FR words. TSS is then used to match code classes with NFRs (IDF(w_i) in Eq.(13) is calculated considering source code classes as system artifacts).

Table 11 Splitting the variable `requesthcprecord`

Split	Action	Output
`r equesthcprecord`	no-split	
`re questhcprecord`	no-split	
`req uesthcprecord`	no-split	
`requ esthcprecord`	no-split	
`reque sthcprecord`	no-split	
`reques thcprecord`	no-split	
`request hcprecord`	split	`request`
`h cprecord`	no-split	
`hc precord`	no-split	
`hcp record`	split	`record`
`hcp`	no-split	`hcp`

The performance of TSS, LSI, and VSM is measured in terms of precision and recall. Recall measures the percentage of correct links that are retrieved, and precision measures the percentage of retrieved links that are correct. Formally, if A is the set of correct links and B is the set of retrieved links, then recall ($\boldsymbol{R}$) and precision ($\boldsymbol{P}$) can be defined as:

$$R = |A \cap B| / |A| \tag{16}$$

$$P = |A \cap B| / |B| \tag{17}$$

To prepare our answer sets, the original developers from each of our experimental systems were asked to separately identify the individual code classes implementing the different NFRs of each system. A single class might implement multiple NFRs, and an NFR might be implemented in multiple classes. Answers were manually inspected and compiled by the researchers using the same data collection protocol in Sect. 5. In cases of conflicts (i.e., whether a certain class implements a certain NFR), the researchers intervened to resolve the issue. Researchers involvement was limited to initiating further discussion with the developers to elaborate on why they thought a certain class was implementing, or not-implementing, a certain NFR. Overall, only a few conflicts were detected and a consensus was reached in all cases during one discussion session.

6.3 Results and discussion

VSM, TSS, and LSI are used to match each NFR word cluster with each class profile in each of our experimental systems. Retrieved classes with similarity scores greater than zero are displayed in a list format. Classes with higher similarity scores appear at the top of the list. The

performance of each IR method over each experimental system is reported over multiple threshold levels $\{0.1, 0.2, \ldots, 1\}$ (at threshold 0.1 only the top 10 % of the retrieved classes in the ranked list of candidate links are considered). In particular, for each system and each IR method, each individual NFR is traced to all the classes in the system using that IR method. Results in terms of precision and recall are then measured at the different thresholds. For each system, the precision and recall values for each IR are averaged for all NFRs at each threshold level, producing the precision-recall performance line of that IR method over the system as shown in Fig 8.

In general, our results show that VSM tends to have higher precision. However, it exhibits the lowest overall average recall in all three systems. In contrast, TSS and LSI are able to achieve higher recall values at different threshold levels. The higher recall levels of TSS in comparison with VSM can be explained based on TSS's ability to capture semantically similar words in the system. In particular, relying solely on lexical matching often leads to information loss [45, 60]. This can be attributed to the fact that, as projects evolve, new and inconsistent terminology gradually finds its way into the system's vocabulary, causing various types of related system artifacts (requirements, source code, design cases) to exhibit a large degree of variance in their lexical contents [5]. This phenomenon, known as the *vocabulary mismatch* problem, is regarded as one of the principal causes of poor accuracy in traceability engines [45]. Therefore, by matching words based on their meanings, TSS is expected to help bridging the textual gap between different system's artifacts, thus improve the retrieval recall.

LSI also seems to be able to capture semantic relations in the system, achieving comparable results to TSS, especially at lower threshold levels. However, the main disadvantage of LSI in comparison with TSS lies in its computationally expensive calibration process. In particular, as observed in LSA analysis (Tables 4, 5, 6, 7), no single universal k works for all systems, rather each system has to be tuned individually to discover the best dimensionality reduction configurations. Such configurations have to be further readjusted for the same system under different tasks. Furthermore, every time k value is changed, the SVD space has to be recalculated. In particular, after an LSA space has been established, changing k values requires recomputing the approximation of the original matrix. However, for methods that rely solely on word counts, even though the co-occurrence matrix has to be updated to accommodate new words, no additional processing over the matrix is required other than a straightforward tabulation of new co-occurrence information.

The main disadvantage of TSS is the fact that unnatural words extracted from source code are ignored. In particular, TSS uses Wikipedia to calculate the similarity of natural language words. Therefore, code abbreviations and acronyms are not included in the analysis. However, a method such as LSI analyzes the internal semantics of the corpus, thus a similarity relation between an acronym and a natural language word might be detected. A proposed solution for this problem is to use abbreviation expansion techniques. Such techniques attempt to automatically expand short forms of source code identifiers (e.g., abbreviations and acronyms) into their long forms, consisting mainly of natural words. However, such techniques are still far from achieving optimal accuracy [43, 71]. Nonetheless, we believe that a future study dedicated to investigating the effect of such techniques on retrieval accuracy is worth pursuing.

The low precision of all methods in all three systems can be explained by the impurity of our NFR word clusters. In particular, due to the near-optimal clustering process, some NFR clusters contain noise, or unrelated words, that contribute to the similarity calculations, thus enhancing the possibility of retrieving unrelated code classes. This problem is more dominant in LSI which seems to achieve drastically lower precision than TSS and VSM especially at

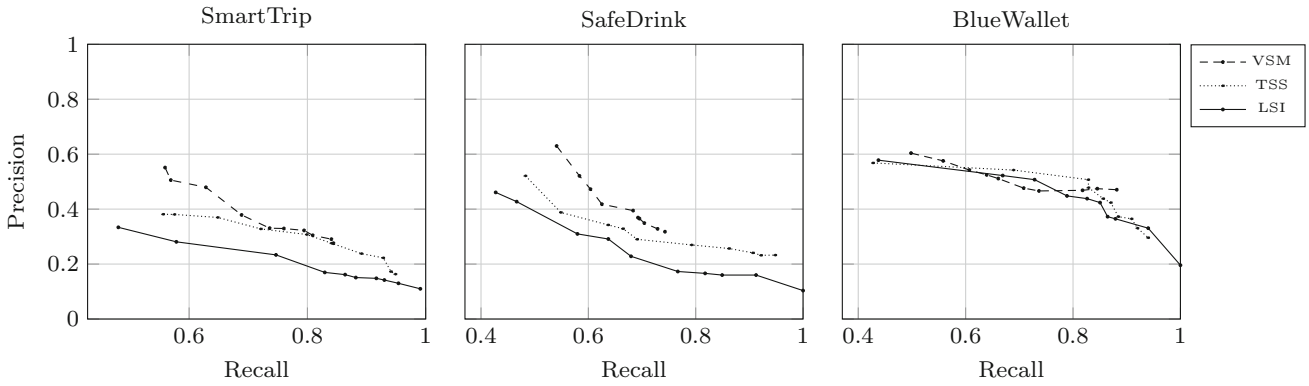

Fig. 8 The performance of VSM, LSI, and TSS in recovering NFR traceability links in terms of recall and precision at different threshold levels

lower thresholds. Another observation is that all methods seem to be consistently achieving a better performance in the *BlueWallet* system. Further investigation revealed that this improvement in the performance can be attributed to the fact that this particular system is larger in size (lines of code and comments) than the other two systems (Fig. 9). Therefore, more data are available for the different IR methods to work with. For instance, a technique such as TSS, which relies on natural language words, benefits greatly from more code comments which are considered a very rich source of English words. LSI is also a data-intensive techniques that tends to perform better with larger amounts of data [1].

Table 12 shows the number of classes in each system implementing each NFR and the F_2 values achieved by each IR method over each experimental system for each individual NFR. These results are measured at a 70 % threshold level, a heuristic that is commonly used to report results in traceability studies [26]. The results in Table 12 show that some NFRs were easier to trace than others. For instance, *BlueWallet* enforces a strong *precision* NFR. This NFR is implemented in 241 classes in the system. The developers of *BlueWallet* indicated that each class that did some sort of calculations had to enforce this constraint in order to ensure accurate results. Almost each of these classes has the words *accuracy*, *precision*, *calculation*, *margin* and *decimal*. Therefore, our IR methods had no problem recovering the *precision* NFR traceability links. Similarly, *SafeDrink* enforces strong *safety* and *privacy* NFRs. These two constraints are implemented in 43 and 31 classes, respectively. These classes contain a large number of safety and privacy semantically related words especially in their code comments. Therefore, a method such as TSS was able to recover these classes with high accuracy levels. In contrast, some other NFRs were more challenging to trace. For instance, the *privacy* NFR in *SmartTrip* is implemented in 7 classes. Our results show that VSM, LSI, and TSS all managed to retrieve and rank all these classes (100 % recall) at the 70 % threshold level. However, so many other unrelated classes (false positives) were also retrieved, taking the precision down to 21, 6, and 6 % in VSM, TSS, and LSI, respectively. This decline in the precision can be attributed to the fact that words representing this particular NFR in *SmartTrip* were semantically vague, thus linking its cluster to several other unrelated classes.

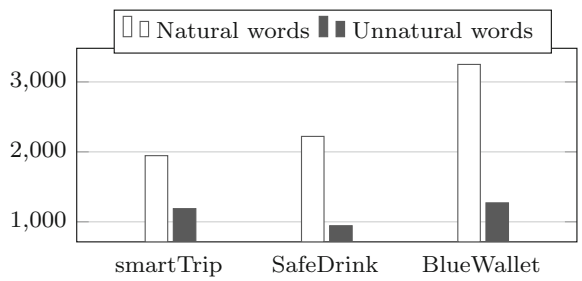

Fig. 9 The number of natural and unnatural words extracted from each of our experimental systems

In summary, to answer **RQ3**, TSS seems to be achieving a balance in accuracy between VSM and LSI. Given that recall is often emphasized over precision in traceability research [45], TSS can be considered as a more effective approach for NFR traceability link recovery.

7 Experimental limitations

The experiment presented in this paper has several limitations that might affect the validity of the results. This section discusses these limitations in terms of external, internal, and construct validities [28].

7.1 External validity

The experiment presented in this paper has several limitations that might affect the external validity of the results. In particular, our results 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 our analysis. In particular, our experimental dataset 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. Another limitation is the fact that all these experimental systems are object-oriented systems that are developed in Java. Thus, whether our findings will generalize over structured code, or systems developed in different programming languages, is still unclear.

Certain design decisions might also impact the external validity of our results. For example, we only experimented with a limited number of clustering algorithms (hierarchical and partitioning) and similarity methods (co-occurrence, thesaurus-based, and latent semantics). In terms of traceability, only TSS, VSM, and LSI were considered in our analysis. Therefore, it is still unclear whether other similarity measures, clustering algorithms, or IR methods (latent dirichlet allocation (LDA) [60, 69] or Jensen-Shannon [1]) might be more appropriate for our approach. Nonetheless, we believe that using this combination of methods that are broadly used in related research is sufficient for drawing preliminary conclusions.

Table 12 NFR traceability results for each individual NFR

NFR	*SmartTrip*				*SafeDrink*				*BlueWallet*			
	N	VSM	TSS	LSI	N	VSM	TSS	LSI	N	VSM	TSS	LSI
security	4	0.43	0.17	0.41	31	0.69	0.41	0.41	46	0.77	0.89	0.88
performance	42	0.78	0.77	0.38	7	0.38	0.42	0.59	14	0.75	0.44	0.47
accessibility	4	0.69	0.41	0.45	32	0.57	0.45	0.34	3	0.67	0.6	0.59
accuracy	24	0.52	0.57	0.57	14	0.65	0.61	0.33	241	0.93	0.73	0.73
portability	31	0.49	0.56	0.69	24	0.71	0.71	0.31	1	0	0.83	0.83
safety	12	0.58	0.65	0.21	43	0.75	0.67	0.35	NA	NA	NA	NA
legal	3	0.29	0.42	0.08	29	0.63	0.54	0.52	17	0.81	0.59	0.59
privacy	7	0.57	0.24	0.24	31	0.85	0.64	0.43	12	0.67	0.9	0.91
reliability	28	0.66	0.79	0.46	3	0.29	0.65	0.88	26	0.63	0.5	0.33
availability	4	0.64	0.52	0.3	NA	NA	NA	NA	9	0.54	0.48	0.41
interoperability	39	0.75	0.32	0.53	3	0.23	0.29	0	NA	NA	NA	NA

N is the number of classes in the system. VSM, LSI, and TSS columns show the F_2 measure achieved by each IR method in each system at a 70 % threshold level

An external validity threat might stem from the fact that only a limited number of NFRs (Table 9) were included in the analysis. Industry standards, such as the ISO standards 9126 and 25010, list more fine-grained categories of NFR. Including such categories and subcategories in our analysis might change the performance drastically. However, our decision to only consider these general categories of NFRs can be justified based on the fact they categories cover most of the quality constraints enforced in our experimental systems. Therefore, we did not feel the need to introduce more fine-grained categories to avoid confusing our study participants. In particular, considering all categories and subcategories of NFRs will substantially increase the complexity of the study, especially when creating the ground-truth answer set, thus impacting the overall accuracy of the analysis.

7.2 Internal validity

Internal validity refers to factors that might affect the causal relations established in the experiment [28]. A potential threat to the proposed study's internal validity is the fact that human judgment is used to prepare our classification and traceability answer sets. This might result in an experimental bias as different project personnel might have different understandings 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 and trace each requirement.

The class-granularity level adopted in our analysis can also influence the performance. In particular, different granularity levels might considerably change the behavior of methods such as LSI and PMI. Therefore, we believe that a future study (e.g., [10]) dedicated to investigating the effect of granularity level on the performance of semantic similarity methods and clustering algorithms is necessary to further confirm our observations.

An internal validity argument could be made about using a brute-force strategy to calibrate methods such as LSI or to configure different clustering algorithms. For instance, for larger datasets, other automated optimization strategies might be more computationally efficient [55]. However, since our data space is relatively limited in size, exhaustive search solutions can be computationally feasible. Furthermore, a brute-force strategy is highly likely to find a solution if a solution exists in the space.

7.3 Construct validity

Construct validity is the degree to which the various performance variables accurately measure the concepts they purport to measure [28]. In our experiment, there are minimal threats to construct validity as standard performance measures (e.g., precision, recall, cluster quality), which are extensively used in related research, are used to assess the performance of the different investigated methods. We believe that these measures sufficiently capture and quantify the different aspects of the performance we are interested in.

8 Conclusions, applications, and future work

This section summarizes our main findings in this paper, discusses the potential practical significance of the proposed approach, and explores directions for future work.

8.1 Summary

This paper proposes a novel, unsupervised, and computationally efficient approach for detecting, classifying, and tracing non-functional software requirements (NFRs). The analysis in this paper is based on the main assumption that NFRs tend to be implicitly enforced by the software system's functional requirements. Such knowledge can be captured, modeled, and used to identify specific NFRs in the system.

In particular, a comprehensive systematic analysis of a series of word semantic similarity methods and clustering algorithms was conducted to identify the clustering configurations that generate the most cohesive groups of functional requirement words. A semantically aware cluster quality function was designed to capture the notion of semantic coherence of generated word clusters. Three software systems from different application domains were used to conduct our analysis.

The results showed that hierarchical clustering algorithms, especially average linkage (AL), were more effective in generating thematic clusters of FR words than partitioning algorithms (k-medoids). In terms of semantic distance, our results showed that the method that exploited the online encyclopedia Wikipedia was more accurate than methods that relied on linguistic dictionaries or the corpus itself for semantic similarity analysis. In general, from a linguistic point of view, Wikipedia provides a balance in size, quality, and structure between the highly structured, but limited in coverage, linguistic databases (e.g., WordNet), and the large-scale, but less-structured corpora such as the entire Web. Furthermore, our results showed that a corpus-based method such as PMI suffered from the lack of sufficient textual data in our systems. LSA, on the other hand, was able to achieve competitive results. However, the computationally expensive calibration process associated with this technique can limit its practicality.

Generated clusters of FR words were classified into different categories of NFRs based on their semantic similarity to basic NFR labels. The TSS measure of textual semantic similarity was then used to assign FRs to different categories of NFRs. Our analysis showed that a semantic similarity threshold in the range of [0.2–0.4] generated the best classification results. Lower similarity thresholds resulted in a very high recall, but very low precision, while higher similarity thresholds caused a drastic drop in the recall (very low classification rate).

In terms of traceability, three IR methods, exploiting different lexical and semantic schemes, were used to match the discovered NFRs with code classes implementing them. To enhance the retrieval accuracy, a novel algorithm was proposed to extract natural language words from source code identifiers. The results showed that relying solely on lexical matching often led to information loss. On the contrast, methods that utilized the textual semantics of software words (e.g., TSS and LSI) were found to be more successful in terms of recall and precision. LSI, however, achieved a very low precision at higher threshold levels, unlike TSS, which was found to achieve a balance between precision and recall, thus providing a computationally efficient alternative that can be adequate for practical IR-based traceability solutions. In summary, the proposed approach (Fig. 1) can be described as a formal procedure as follows:

Algorithm 1 NFR classification and traceability

1: **procedure**
2: Extract words from the system's FRs
3: Calculate NGD_{Wiki} similarity between FR words
4: Cluster FR words using AL based on their NGD_{Wiki}
5: Find the number of clusters using Q - Eq.(11)
6: Classify FR word clusters to NFR types using Eq.(12)
7: Assign FRs to NFRs using Eq.(13)
8: Trace NFRs to code classes using TSS - Eq.(13)
9: **end procedure**

8.2 Applications, reflection, and future work

There is no doubt that there is still a major gap in the state of research and practice between functional and non-functional software requirements. In general, majority of requirement documentation, modeling, and traceability research is focused on functional requirements, with only a little attention paid to non-functional requirements. This can be attributed to the general lack of understanding of the important role of NFRs and their long-term influence on the different phases of the project's life cycle. The work presented in this paper, as well as previous work in this domain [22, 24, 46, 78], represents a step toward bridging this gap.

The approach presented in this paper relies on the online collaborative encyclopedia Wikipedia to estimate the semantic similarity between individual FR words. This provides a practical advantage over other techniques that rely on static linguistic dictionaries or *ad-hoc* lists of words for classification. In particular, the textual content of Wikipedia evolves constantly in such a way that is aligned with the way natural language evolves in society [34, 82]. Therefore, even if the language of the project, used in requirements or code, has changed over time, the proposed approach should be able to adapt to these changes without the need for updating any static dictionaries or retraining any classifiers.

In terms of classification quality, our analysis showed an average classification recall of 83 %. As for precision, our approach managed to keep a precision average of 53 % in

our three experimental systems. As implied earlier, in our analysis we emphasized recall over precision, this decision was adopted based on the previous observation that false positives can be easier to deal with than false negatives. For example, an incorrectly classified FR can be simply ignored by the user. However, an FR that enforces a certain NFR but was not classified under that NFR can be hard to find.

Our findings in the traceability phase of our analysis emphasize the importance of enforcing a standard coding convention in software institutions. In particular, using meaningful code identifiers not only enhances the readability and maintainability of the code, but also improves the performance of tools that rely on such information for operation. Similarly, comments can be an essential source of semantic knowledge as they are expressed mainly in natural language. Our findings in this paper help to highlight these long-term benefits of promoting such culture in software development.

The proposed approach in this paper is unsupervised. In particular, it does not require any training and can be operated with minimum calibration. This design decision was enforced to facilitate an easy transfer of our findings to practice. In other words, simplify the process of implementing the approach and producing the output while keeping the operating cost as low as possible [54]. Finally, The line of research in this paper has opened several research directions to be pursued in our future work. These directions can be described as follows:

- Empirical analysis: Our experimental dataset will be enhanced with more experimental systems. In addition, more participants will be recruited to help to prepare larger answer sets. In terms of methods, we will continue to explore different types of word semantic similarity methods and clustering techniques that exploit different aspects of software artifacts to generate more coherent software word clusters.
- Applications: Our future research will be focused on utilizing discovered NFRs and their traceability links in basic software engineering tasks, such as code design and architecture, software verification and validation, and change impact analysis.
- Tool support: A set of working prototypes that implement our findings in this paper will be developed. These prototypes will include stand-alone tools and plug-ins that developers can use as an integral part of their daily software production practices. Furthermore, working prototypes will allow us to conduct long-term usability studies to gain a better understanding of our approach's scalability, usability, and scope of applicability.

Acknowledgments The authors would like to thank our study participants and the Institutional Review Board (IRB) at LSU for approving this research. This work was supported in part by the Louisiana Board of Regents Research Competitiveness Subprogram (LA BoR-RCS), contract number: LEQSF(2015-18)-RD-A-07.

References

1. Abadi A, Nisenson M, Simionovici Y (2008) A traceability technique for specifications. In: International conference on program comprehension, pp 103–112
2. Aggarwal C, Zhai C (2012) A survey of text clustering algorithms. Mining text data. Springer, Newyork, pp 77–128
3. Anquetil N, Fourrier C, Lethbridge T (1999) Experiments with clustering as a software remodularization method. In: Working conference on reverse engineering, pp 235–255
4. Anquetil N, Lethbridge T (1998) Assessing the relevance of identifier names in a legacy software system. In: Conference of the centre for advanced studies on collaborative research, pp 4–14
5. Antoniol1 G, Guéhéneuc Y, Merlo E, Tonella P (2007) Mining the lexicon used by programmers during software evolution. In: International conference on software maintenance, pp 14–23
6. Arthur D, Vassilvitskii S (2007) K-means++: the advantages of careful seeding. In: Annual ACM-SIAM symposium on discrete algorithms, pp 1027–1035
7. Bekkerman R, El-Yaniv R, Tishby N, Winter Y (2003) Distributional word clusters vs. words for text categorization. J Mach Learn Res 3:1183–1208
8. Bollegala D, Matsuo Y, Ishizuka M (2007) Measuring semantic similarity between words using web search engines. In: International conference on world wide web, pp 757–766
9. Budiu R, Royer C, Pirolli P (2007) Modeling information scent: a comparison of LSA, PMI and GLSA similarity measures on common tests and corpora. In: Large scale semantic access to content (text, image, video, and sound), pp 314–332
10. Bullinaria J, Levy J (2007) Extracting semantic representations from word co-occurrence statistics: a computational study. Behav Res Methods 39(3):510–526
11. van Rijsbergen CJ (1979) Information retrieval. Butterworths, New York
12. Carreño G, Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: International conference on software engineering, pp 343–348
13. Casamayor A, Godoy D, Campo M (2010) Identification of non-functional requirements in textual specifications: a semi-supervised learning approach. Inf Softw Technol 52(4):436–445
14. Chang J, Boyd-Graber J, Gerrish S, Wang C, Blei D (2009) Reading tea leaves: how humans interpret topic models. Curran Associates, County Down, pp 288–296
15. Chen J, Ren Y, Riedl J (2010) The effects of diversity on group productivity and member withdrawal in online volunteer groups. In: SIGCHI conference on human factors in computing systems, pp 821–830
16. Chung L, do Prado Leite J (2009) On non-functional requirements in software engineering. Concept Model Found Appl Lecture Notes Comput Sci 5600:363–379
17. Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. Kluwer Academic, Boston
18. Church K, Hanks P (1990) Word association norms, mutual information, and lexicography. Comput Ling 16(1):22–29
19. Cilibrasi R, Vitanyi P (2007) The Google similarity distance. IEEE Trans Knowl Data Eng 19(3):370–383

20. Cleland-Huang J, Chang C, Christensen M (2003) Event-based traceability for managing evolutionary change. IEEE Trans Softw Eng 29(9):796–810
21. Cleland-Huang J, Heimdahl M, Huffman-Hayes J, Lutz R, Mäder P (2012) Trace queries for safety requirements in high assurance systems. In: International conference on requirements engineering: foundation for software quality, pp 179–193
22. Cleland-Huang J, Schmelzer D (2003) Dynamically tracing non-functional requirements through design pattern invariants. In: Workshop on traceability in emerging forms of software tracing non-functional requirements
23. Cleland-Huang J, Settimi R, BenKhadra O, Berezhanskaya E, Christina S (2005) Goal-centric traceability for managing non-functional requirements. In: International conference on software engineering, pp 362–371
24. Cleland-Huang J, Settimi R, Zou X, Solc P (2007) Automated classification of non-functional requirements. Requir Eng 12(2):103–120
25. Cysneiros LM (2007) Evaluating the effectiveness of using catalogues to elicit nonfunctional requirements. In: Workshop em Engenharia de Requisitos, pp 107–115
26. De Lucia A, Oliveto R, Sgueglia P (2006) Incremental approach and user feedbacks: a silver bullet for traceability recovery. In: International conference on software maintenance, pp 299–309
27. De Lucia A, Oliveto R, Tortora G (2009) Assessing IR-based traceability recovery tools through controlled experiments. Empir Softw Eng 14(1):57–92
28. Dean A, Voss D (1999) Design and analysis of experiments. Springer, New York
29. Deerwester S, Dumais S, Furnas G, Landauer T, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407
30. Deißenböck F, Pizka M (2005) Concise and consistent naming. In: International workshop on program comprehension, pp 97–106
31. Demmel J, Kahan W (1990) Accurate singular values of bidiagonal matrices. J Sci Stat Comput 11(5):873–912
32. Fellbaum C (1998) WordNet: an electronic lexical database. MIT Press, Cambridge
33. Funahashi T, Yamana H (2010) 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, pp 114–125
34. Gabrilovich E, Markovitch S (2007) Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In: International joint conference on artificial intelligence, pp 1606–1611
35. Glinz M (2007) On non-functional requirements. In: IEEE international requirements engineering conference, pp 21–26
36. Goldin L, Berry D (1997) AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Autom Softw Eng 4(4):375–412
37. Gotel O, Cleland-Huang J, Huffman-Hayes J, Zisman A, Egyed A, Grnbacher P, Dekhtyar A, Antoniol G, Maletic J (2012) The grand challenge of traceability (v1.0). In: Software and systems traceability. Springer, London
38. Gracia J, Trillo R, Espinoza M, Mena E (2006) Querying the web: a multiontology disambiguation method. In: International conference on web engineering, pp 241–248
39. Gross D, Yu E (2000) From non-functional requirements to design through patterns. Requir Eng 6(1):18–36
40. Guo W, Li H, Ji H, Diab M (2013) Linking tweets to news: a framework to enrich short text data in social media. In: Annual meeting of the association for computational linguistics, pp 239–249
41. Hearst M, Pedersen J (1996) Reexamining the cluster hypothesis: scatter/gather on retrieval results. In: International ACM SIGIR conference on Research and development in information retrieval, pp 76–84
42. Hill E, Binkley D, Lawrie D, Pollock L, Vijay-Shanker K (2014) An empirical study of identifier splitting techniques. Empir Softw Eng 19(6):1754–1780
43. Hill E, Fry Z, Boyd H, Sridhara G, Novikova Y, Pollock L, Vijay-Shanker K (2008) Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools. In: International working conference on mining software repositories, pp 79–88
44. Holzinger A, Yildirim P, Geier M, Simonic KM (2013) Quality-based knowledge discovery from medical text on the web. In: Pasi G, Bordogna G, Jain L (eds) Quality issues in the management of web information. Springer, Berlin, pp 145–158
45. Huffman-Hayes J, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans Softw Eng 32(1):4–19
46. Kassab M, Ormandjieva O, Daneva M (2009) A metamodel for tracing non-functional requirements. In: World congress on computer science and information engineering, pp 687–694
47. Kaufman L, Rousseeuw P (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York
48. Kotonya G, Sommerville I (1998) Requirements engineering: processes and techniques. Wiley, New York
49. Kuhn A, Ducasse S, Gîrba T (2007) Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3):230–243
50. Landauer T, Dutnais S (1997) A solution to plato's problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge. Psychol Rev 104(2):211–240
51. Lau J, Newman D, Karimi S, Baldwin T (2010) Best topic word selection for topic labelling. In: International conference on computational linguistics, pp 605–613
52. Leacock C, Chodorow M (1998) Combining local context and WordNet similarity for word sense identification. MIT Press, Cambridge
53. Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: Annual international conference on systems documentation, pp 24–26
54. Lo D, Nagappan N, Zimmermann T (2015) How practitioners perceive the relevance of software engineering research. In: Joint meeting on foundations of software engineering, pp 415–425
55. Lohar S, Amornborvornwong S, Zisman A, Cleland-Huang J (2013) Improving trace accuracy through data-driven configuration and composition of tracing features. In: Joint meeting on foundations of software engineering, pp 378–388
56. Luisa M, Mariangela F, NoviInverardi P (2004) Market research for requirements analysis using linguistic tools. Requir Eng 9(1):40–56
57. Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behav Res Methods Instrum Comput 28(2):203–208
58. Maalej W, Nabil H (2015) Bug report, feature request, or simply praise? On automatically classifying app reviews. In: Requirements engineering conference, pp 116–125
59. Mahmoud A (2015) An information theoretic approach for extracting and tracing non-functional requirements. In: International requirements engineering conference
60. Mahmoud A, Niu N (2015) On the role of semantics in automated requirements tracing. Requir Eng 20(3):281–300
61. Mihalcea R, Corley C, Strapparava C (2006) Corpus-based and knowledge-based measures of text semantic similarity. In: National conference on artificial intelligence, pp 775–780

62. Mimno D, Wallach H, Talley E, Leenders M, McCallum A (2011) Optimizing semantic coherence in topic models. In: The conference on empirical methods in natural language processing, pp 262–272
63. Mirakhorli M, Cleland-Huang J (2012) Tracing non-functional requirements. In: Zisman A, Cleland-Huang J, Gotel O (eds) Software and systems traceability. Springer, New York, pp 299–320
64. Mylopoulos J, Chung L, Nixon B (1992) Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans Softw Eng 18(6):483–497
65. Newman D, Han Lau J, Grieser K, Baldwin T (2010) Automatic evaluation of topic coherence. In: Annual conference of the North American chapter of the association for computational linguistics, pp 100–108
66. Newman D, Noh Y, Talley E, Karimi S, Baldwin T (2010) Evaluating topic models for digital libraries. In: Annual joint conference on digital libraries, pp 215–224
67. Niu N, Mahmoud A (2012) Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited. In: IEEE international requirements engineering conference, pp 81–90
68. Nuseibeh B (2001) Weaving together requirements and architectures. Computer 34(3):115–119
69. Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2010) On the equivalence of information retrieval methods for automated traceability link recovery. In: International conference on program comprehension, pp 68–71
70. Peraldi Frati MA, Albinet A (2010) Requirement traceability in safety critical systems. In: Workshop on critical automotive applications: robustness and safety, pp 11–14
71. Pollock L (2012) Leveraging natural language analysis of software: achievements, challenges, and opportunities. In: IEEE international conference on software maintenance, pp 4–4
72. Pollock L, Vijay-Shanker K, Hill E, Sridhara G, Shepherd D (2013) Natural language-based software analyses and tools for software maintenance, Lecture notes in computer science, vol 7171. Springer, Berlin, pp 94–125
73. Porter F (1997) An algorithm for suffix stripping. Morgan Kaufmann Publishers Inc, Burlington
74. Resnik P (1995) Using information content to evaluate semantic similarity in a taxonomy. In: International joint conference on artificial intelligence, pp 448–453
75. Rosario B (2000) Latent semantic indexing: an overview. INFOSYS 240 Spring Paper, University of California, Berkeley
76. Salton G, Wong A, Yang C (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
77. Sawyer P, Rayson P, Cosh K (2005) Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans Softw Eng 31(11):969–981
78. Slankas J, Williams L (2013) Automated extraction of non-functional requirements in available documentation. In: International workshop on natural language analysis in software engineering (NaturaLiSE), pp 9–16
79. Slonim N, Tishby N (2000) Document clustering using word clusters via the information bottleneck method. In: International ACM SIGIR conference on research and development in information retrieval, pp 208–215
80. Sousa D, Sarmento L, Rodrigues EM (2010) Characterization of the twitter replies network: are user ties social or topical? In: International workshop on search and mining user-generated contents, pp 63–70
81. Sridhara G, Hill E, Pollock L, Vijay-Shanker K (2008) Identifying word relations in software: A comparative study of semantic similarity tools. In: IEEE international conference on program comprehension, pp 123–132
82. Strube M, Ponzetto S (2006) Wikirelate! computing semantic relatedness using Wikipedia. In: National conference on artificial intelligence, pp 1419–1424
83. Thelwall M (2008) Extracting accurate and complete results from search engines: case study windows live. J Am Soc Inform Sci Technol 59(1):38–50
84. Turney P (2001) Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In: European conference on machine learning, pp 491–502
85. Woon WL, Madnick S (2009) Asymmetric information distances for automated taxonomy construction. Knowl Inf Syst 21(1):91–111
86. Wu Z, Palmer M (1994) Verbs semantics and lexical selection. In: Annual meeting on association for computational linguistics, pp 133–138
87. Xiang Z, Wöber K, Fesenmaier D (2008) Representation of the online tourism domain in search engines. J Travel Res 47(2):137–150
88. Zhang W, Yang Y, Wang Q, Shu F (2011) An empirical study on classification of non-functional requirements. In: International conference on software engineering and knowledge engineering, pp 190–195