Automated Extraction and Clustering of Requirements Glossary Terms

Chetan Arora, Mehrdad Sabetzadeh, *Member, IEEE,* Lionel Briand, *Fellow, IEEE,* and Frank Zimmer, *Member, IEEE*

Abstract—A glossary is an important part of any software requirements document. By making explicit the technical terms in a domain and providing definitions for them, a glossary helps mitigate imprecision and ambiguity. A key step in building a glossary is to decide upon the terms to include in the glossary and to find any related terms. Doing so manually is laborious, particularly for large requirements documents.

In this article, we develop an automated approach for extracting candidate glossary terms and their related terms from natural language requirements documents. Our approach differs from existing work on term extraction mainly in that it *clusters* the extracted terms by relevance, instead of providing a flat list of terms. We provide an automated, mathematically-based procedure for selecting the number of clusters. This procedure makes the underlying clustering algorithm transparent to users, thus alleviating the need for any user-specified parameters.

To evaluate our approach, we report on three industrial case studies, as part of which we also examine the perceptions of the involved subject matter experts about the usefulness of our approach. Our evaluation notably suggests that: (1) Over requirements documents, our approach is more accurate than major generic term extraction tools. Specifically, in our case studies, our approach leads to gains of 20% or more in terms of recall when compared to existing tools, while at the same time either improving precision or leaving it virtually unchanged. And, (2) the experts involved in our case studies find the clusters generated by our approach useful as an aid for glossary construction.

Index Terms—Requirements Glossaries, Term Extraction, Natural Language Processing, Clustering, Case Study Research.

1 Introduction

A requirements glossary makes explicit and provides definitions for the salient terms in a requirements document. A requirements glossary may further provide information about the synonyms, related terms, and example usages of the salient terms. The stakeholders of a system may consult the requirements glossary for various reasons, e.g., to familiarize themselves with the technical terms in the system's domain, to learn about everyday words that may have a specific meaning in the system, to look up abbreviations and related terms, and to understand terminology variations.

Glossaries enhance the requirements engineering process in a number of ways [1], [2], [3], [4]. In particular, glossaries (1) improve the accuracy and understandability of requirements, (2) mitigate vagueness and ambiguity, (3) promote better communication between the stakeholders by establishing a common and consistent language, and (4) provide a stepping stone for further requirements elaboration, e.g., through data and process modeling. The lack of a requirements glossary can hinder teamwork and potentially jeopardize the success of a project [5].

In general and to maximize the benefits mentioned above, it is recommended that a glossary should be built at the same time as when the requirements are being specified [3], [4]. Doing so, however, is not always feasible due to time pressures. Too much upfront investment into the glossary may also be an issue from a cost-effectiveness standpoint, e.g., when the requirements are volatile and expected to change significantly as they are refined and prioritized.

Consequently, requirements glossaries may be built *after the fact* and only when the requirements have sufficiently stabilized. This situation is, for example, reflected in the industrial requirements that are the subject of the case studies in this article (Section 5). To build a glossary after the fact, the terms to include in the glossary need to be extracted from the underlying documents. For large documents, a manual extraction of the terms may require a significant amount of effort, thus leaving less human resources for more complex tasks, e.g., writing the definitions for the glossary terms.

Our objective in this article is to automatically *extract* candidate glossary terms from natural language requirements and organize these terms into *clusters* of related terms. We illustrate the process using the example of Fig. 1. In Fig. 1(a), we show the requirements for which a glossary needs to be built. The requirements concern a satellite software component and represent a small fraction of a larger requirements document. The full document is the subject of one of the case studies in this article, as we discuss later. To protect confidentiality, the requirements have been sanitized without affecting their substance or structure. The abbreviations "GSI", "STS", and "DB" in the requirements stand for "Ground Station Interface", "Surveillance and Tracking System", and "Database", respectively.

Given the requirements in Fig. 1(a), we would like to first obtain a set of candidate terms such as those in Fig. 1(b), and then group these terms into clusters such as those in Fig. 1(c). By bringing together the related terms, these clusters

- *C. Arora, M. Sabetzadeh, and L. Briand are with the SnT Centre for Security, Reliability, and Trust, University of Luxembourg, Luxembourg, L-2721. E-mail: {chetan.arora, mehrdad.sabetzadeh, lionel.briand}@uni.lu*
- *F. Zimmer is with SES Techcom, Luxembourg. E-mail: frank.zimmer@ses.com*

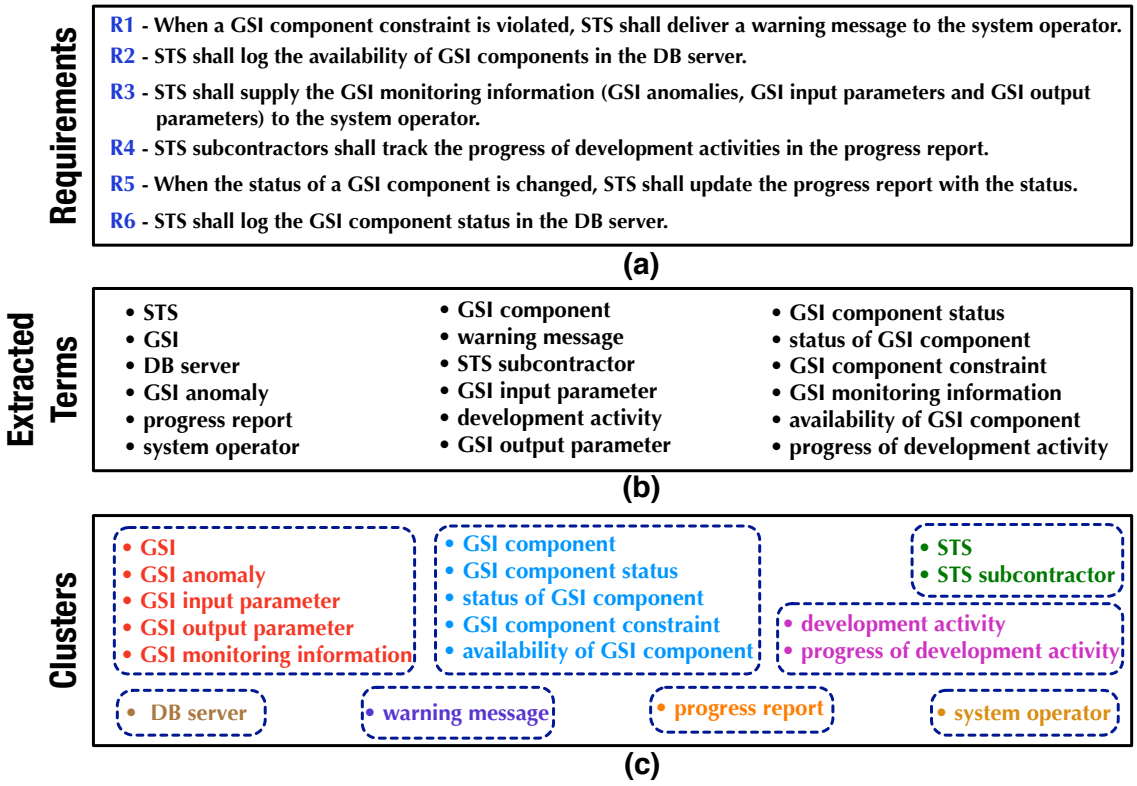


Fig. 1. (a) Example requirements from a satellite component, (b) candidate glossary terms extracted from the requirements, (c) candidate terms organized into clusters.

can provide assistance to the analysts in a number of tasks, including deciding about the terms to include in the glossary, writing definitions for the glossary terms, and identifying potential consistency issues such as the use of variant phrases for referring to the same concept. An example of variant phrases in Fig. 1 is "GSI component status" and "status of GSI component".

Our work fits most closely with existing work on term extraction, which deals with automatic identification of the terminology in a given text corpus [6], [7]. Many strands of work exist on the subject, e.g., [8], [9], [10], [11], to note a few. Despite term extraction being widely studied, existing tools are not tailored to requirements documents.

An important limitation that we have observed in generic term extraction tools is that these tools, when applied to requirements documents, yield poor recall, i.e., they miss a considerable number of glossary terms. This limitation is partly explained by filtering heuristics that are not suited to requirements documents. An example filtering heuristic is the exclusion, from the candidate glossary terms, of terms that are infrequent. While this heuristic is often necessary for extracting terms from large heterogeneous corpora, e.g., collections of books or articles, the heuristic is likely to filter important terms that, despite having a low frequency of appearance, would warrant a precise definition when used in a requirements document. Poor recall is in another part due to the absence of heuristics for combining adjacent phrases under certain conditions. For example, consider the variants "GSI component status" and "status of GSI component", mentioned earlier. One would expect that these variants will be treated the same way by a term extractor. However, the tools we have investigated (Section 2.4) would detect only the former because it is a single noun phrase, but not the latter, because it is a combination of two noun phrases.

We take steps in this article to address the above limitation. More importantly, what contrasts our work from the existing work on term extraction is that, instead of producing a flat list of candidate terms, our approach produces clusters of related terms. As we argue more precisely later in the article based on our empirical results, we believe that clusters provide a more suitable basis than flat lists for performing the tasks related to glossary construction.

1.1 Contributions

We propose an automated solution for extracting and clustering candidate glossary terms in requirements documents. Specifically, we make the following contributions:

(1) We develop a term extraction technique using a well-known natural language processing (NLP) task called *text chunking* [12]. In particular, we are interested in noun phrase (NP) chunks in requirements documents. NPs correspond closely to candidate glossary terms. We propose complementary heuristics to address limitations in a naive application of chunking.

(2) We develop a technique to cluster candidate glossary terms based on syntactic and semantic similarity measures for natural language. An important consideration with regard to clustering is selecting an appropriate number of clusters. To avoid the

need for users to specify this number in an arbitrary manner, we provide automated guidelines for estimating the optimal number of clusters.

(3) We report on the design and execution of three industrial case studies. Through these case studies, we tackle several research questions, including how our term extraction approach compares with the existing generic approaches, how one can best tune clustering for grouping together related terms, and how industry experts perceive the usefulness of the clusters generated by our approach. Our results notably suggest that: Firstly, our term extraction technique outperforms, by a factor of 20% or more, all the generic term extraction tools compared with in terms of recall, while at the same time also outperforming all but one of these tools in terms of precision. Our term extraction technique, when compared with the best generic alternative, has lower precision in two of our case studies. Nevertheless, the precision loss is negligible (0.9% in one case study and 1.2% in the other) and significantly outweighed by gains in recall. Secondly, the experts find the clusters computed by our clustering technique useful as an aid for defining the glossary terms, for identifying the related terms, and for detecting potential terminological inconsistencies.

(4) We develop a tool, named REGICE, implementing our term extraction and clustering techniques. The tool is available at https://sites.google.com/site/svvregice/. To facilitate the replication of our empirical results using REGICE or other tools, we provide on REGICE's website the material for one of our case studies, whose requirements document is public-domain.

This article extends a previous conference paper [13] published at the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'14). In addition to improving the technical aspects of our approach and providing a more detailed exposition of these aspects, this article offers major extensions in the following areas: *(1) Empirical evaluation:* We complement the two case studies in our previous work with a third one (Case-C in Section 5). We provide a more thorough comparison with existing term extraction tools. We consider a larger set of measures for calculating syntactic and semantic similarity. We experiment with three alternative clustering algorithms, rather than just one as in our prior work. And, we systematically examine the opinions of the industry experts involved in our case studies in order to gain insight as to whether our approach is useful in practice. *(2) Tool support:* We have evolved the implementation of our approach into a tool, which is now publicly available. We describe our tool as part of this article.

1.2 Structure of the article

The rest of this article is structured as follows: Section 2 provides background information and compares our work with related work. Section 3 presents our term extraction and clustering techniques. Section 4 describes our tool. Section 5 reports on the design and execution of our case studies. Section 6 discusses threats to validity. Section 7 concludes the article with a summary and directions for future work.

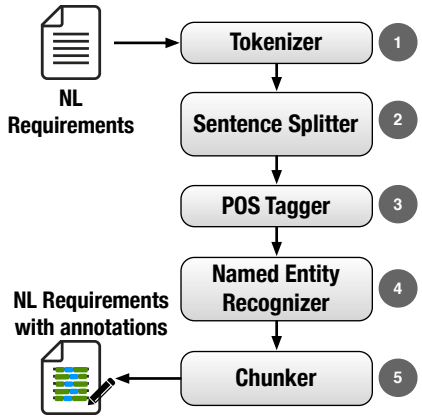


Fig. 2. NLP pipeline for text chunking.

2 Background

This section presents background on the key technologies used in the article. The section further reviews and compares with related strands of work.

2.1 Text Chunking

Our approach builds upon a mature NLP task, known as text chunking. Text chunking is the process of decomposing a sentence into non-overlapping segments [14]. These segments include among others, noun phrases (NPs), prepositional phrases (PPs), and verb phrases (VPs). For the purpose of this article, we are interested only in NPs. An *NP* is a segment that can be the subject or object in a sentence. According to Justeson and Katz [15], NPs account for 99% of the terms in technical glossaries. The remaining 1% are typically VPs. Our term extraction technique does not return any VPs in its results, since the benefits of doing so are significantly outweighed by the overhead of having to manually filter undesirable VPs from the results.

A text chunker is a pipeline of NLP modules running in a sequence over one or more input documents. The generic pipeline for chunking is shown in Fig. 2. The first module in the pipeline is the Tokenizer, which breaks up the input text into tokens. A token can be a word, a number or a symbol. The next module, Sentence Splitter, divides the text into sentences. Subsequently, the POS Tagger annotates each token with a part-of-speech (POS) tag. These tags include, among others, Pronoun, Adjective, Noun, and Verb. Next is the Name Entity Recognizer, where an attempt is made to identify the named entities in the text. Examples of named entities include dates, organizations, URLs, and locations. The main and final step is to identify and mark the text chunks. This is performed by the Chunker module.

Fig. 3 shows the result of running the pipeline of Fig. 2 over requirement R4 from the example in Fig. 1. Once processed by this pipeline, an input document will have annotations for tokens, sentences, parts-of-speech, named entities, and text chunks. As said earlier, of the chunks, we need only the NPs.

The pipeline of Fig. 2 may be realized in many different ways as there are alternative implementations available for

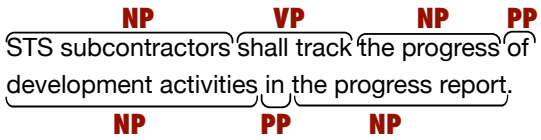


Fig. 3. Text chunking example.

each of the modules in the pipeline. In our previous work [16], we evaluated 144 possible combinations of module implementations for building the text chunking pipeline. In this current article, we do not further evaluate these combinations and instead take one of the most accurate ones identified in our earlier work. The choice of module implementations used is given in Section 4.

2.2 Computing Similarities between Terms

To cluster candidate terms, we need a degree of relatedness between term pairs. We define this degree using syntactic and semantic similarity measures, as we describe next.

2.2.1 Syntactic Similarity Measures

Syntactic (similarity) measures calculate a score for a given pair of terms based on the terms' string content. These measures are usually normalized to a value between 0 and 1, with 0 indicating no similarity at all, and 1 indicating perfect similarity, i.e., string equivalence. For example, syntactic measures would normally yield a high score for the terms "GSI component" and "GSI component status" because of the large textual overlap between the terms. In our empirical evaluation (Section 5), we consider 12 syntactic measures: *Block distance* [17], *Cosine* [17], *Dice's coefficient* [17], *Euclidean distance* [17], *Jaccard* [18], *Char-Jaccard* [18], *Jaro* [18], *Jaro-Winkler* [18], *Level Two (L2) Jaro-Winkler* [18], *Levenstein* [19], *Monge-Elkan* [20], and *SoftTFIDF* [18]. These measures are described in the appendix; see Table A.1.

Syntactic measures can be classified into three categories: *distance-based*, *token-based*, and *corpus-based* [18]. Distance-based measures calculate a score for a given pair of terms by finding the best sequence of edit operations to convert one term into the other. *Levenstein* is an example of such measures. Token-based measures work by treating each term as a bag of tokens and then matching the tokens of different terms. An example such measure is *Cosine*. Corpus-based measures enhance either distance-based or token-based measures by accounting for the characteristics of the corpus from which the terms are drawn. For example, to calculate a similarity score for a pair of terms, *SoftTFIDF* considers the frequency of the terms' constituent tokens in the corpus where the terms appear. The standard implementation of *SoftTFIDF* uses *Jaro-Winkler* (a distance-based measure) as a similarity predicate over tokens.

2.2.2 Semantic Similarity Measures

Given a pair of words, semantic (similarity) measures calculate a similarity score based on the meaning of the words in a dictionary. Similar to syntactic measures, semantic measures

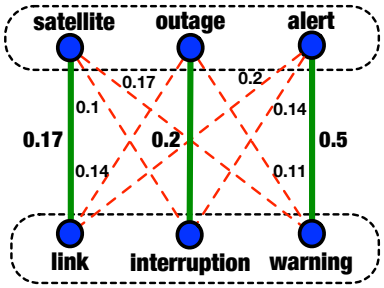


Fig. 4. Example of semantic similarity calculation for multi-word terms.

are often normalized. For example, most semantic measures would produce a non-zero score for the words "communication" and "transmission" because of the semantic affinity between these two words. In this article, we experiment with eight semantic measures: *HSO* [21], *JCN* [22], *LCH* [23], *LIN* [24], *LESK* [25], *PATH* [26], *RES* [27], and *WUP* [28]. These measures are described in the appendix; see Table A.2.

To generalize semantic measures from single-word terms (i.e., tokens) to multi-word terms, one must define a strategy for combining token-level similarity scores. To this end, we adopt the strategy used by Nejati et al. [29]: Given a pair of (multi-word) terms, we treat the terms as bags of tokens. We then calculate similarity scores for all token pairs using the semantic measure of choice. In the next step, we compute an optimal matching of the terms' constituent tokens. A matching is optimal if it maximizes the sum of token-level similarity scores. Finally, we calculate the normalized sum for the optimal matching and take the result as the similarity score for the given terms. More precisely, given a pair (t_1, t_2) of terms, the (term-level) semantic similarity score, $\mathscr{S}(t_1, t_2)$, is:

$$\mathscr{S}(t_1,t_2) = 2 \times \frac{\textit{sum of token similarity scores in optimal match}}{N_1 + N_2}$$

where N_1 and N_2 denote the number of tokens in t_1 and t_2.

Fig. 4 illustrates the calculation of a similarity score for the terms "satellite outage alert" and "link interruption warning". Here, the token-level similarity scores, shown on the lines that connect the tokens, were calculated using the *PATH* measure. The optimal matching between the tokens is shown using solid lines. Based on this optimal matching, the similarity score for the terms in question is: $2 \times (0.17 + 0.2 + 0.5)/(3+3) = 0.29$.

2.3 Clustering

Clustering refers to the task of grouping related objects in a manner that the objects in the same cluster are more similar to one another than to the objects in other clusters [30].

To devise an accurate technique for clustering glossary terms, we need to address two important factors. First, we need to select a suitable clustering algorithm from the alternatives available. Second, we need to define a strategy for tuning the input parameters of the selected algorithm. Having such a strategy is essential in order to avoid the end-user from having to make ad-hoc decisions about the input

parameters. In particular, virtually all clustering algorithms require the number of clusters to be given a priori as an input parameter. Naturally, the value of this parameter varies from one requirements document to another, depending on the document's size and complexity. If a poor choice is made about the number of clusters, the accuracy of clustering may be severely compromised.

Below, we review the clustering algorithms examined in this article as well as the criterion that we use for estimating the optimal number of clusters for a given requirements document.

2.3.1 Clustering Algorithms

We experiment with three well-known clustering algorithms, *K-means*, *Agglomerative Hierarchical* and *EM*, to determine which one(s) are the most accurate in our application context. Our choice of these algorithms is motivated by their prevalent use for clustering of natural-language content [30]. Below, we briefly outline these algorithms. Further details can be found in clustering and data mining textbooks, e.g., see [30].

K-means partitions a given set of data points (in our context, candidate terms) into K clusters, where K is an a-priori-given number. Briefly, K-means attempts to assign each data point to a cluster in a way that maximizes the similarity between the individual data points in each cluster and the center of that cluster, called a *centroid*. The centroids and the cluster membership functions are iteratively improved until convergence, i.e., when a fixpoint is reached. There are alternative methods for initializing the centroids. In this article, we use the baseline version of K-means, where the initial centroids are selected randomly.

(Agglomerative) Hierarchical Clustering groups a set of data points by building a tree-shaped structure, called a *dendrogram*. The data points constitute the dendrogram's leaf nodes. A dendrogram is not one set of clusters, but rather a cluster hierarchy. Each non-leaf node in a dendrogram represents a cluster made up of all leaf nodes (data points) that are descendants of the non-leaf node in question. The algorithm starts by assigning each data point to its own cluster. It then finds the closest pair of clusters, i.e., the pair with the largest similarity, or dually the shortest distance, and merges the cluster pair into one cluster. This process is repeated until all the data points have been absorbed into a single cluster, represented by the root node of the dendrogram. There are several alternatives ways for computing the similarity, or dually, the distance, between two clusters during hierarchical clustering. In this article, we consider eight alternatives: *average link*, *centroid link*, *complete link*, *McQuitty's link*, *median link*, *single link*, *Ward.D link*, and *Ward.D2 link*. A description of these alternatives is provided in the appendix; see Table A.3.

Given a dendrogram, one can obtain a single set of clusters either by cutting the dendrogram at a given height, H, or by splitting the dendrogram into a given number, K, of clusters. Either way, the value of the respective parameter has to be specified by the user. We were unable to identify generalizable guidelines to help decide the value of H in the first option above. In this article, we therefore consider only the second option, i.e., splitting into a prespecified number of clusters.

Expectation Maximization (EM) is a statistical clustering algorithm. In this article, we use a common variation of EM, where it is assumed that a set of observations –in our case, the similarity degrees between candidate terms– is a combination of a given number, K, of multivariate normal distributions [31]. Each distribution, characterized by its mean and covariance matrix, represents one cluster. The EM algorithm attempts to approximate the K individual distributions, so that their combination best fits the observations. Here, K corresponds to the number of clusters. Initially, the EM algorithm chooses random values for the means and covariance matrices of the distributions. The algorithm then iterates through the following two steps until convergence:

– *Expectation step*: Given the means and covariance matrices of the K distributions, estimate the membership probability of each data point (candidate term) in each distribution.

– *Maximization step*: Estimate new values for the means and covariance matrices of the K distributions, using maximum-likelihood estimation [32].

Once the algorithm converges, each data point is assigned to the cluster in which it has the largest membership probability.

2.3.2 Choosing the Number of Clusters

As we stated earlier, choosing an appropriate number of clusters, denoted K in Section 2.3.1, is imperative for the accuracy of clustering. If K is too large, closely related terms will be scattered over different clusters rather than being grouped together; if K is too small, we will be left with clusters in which the terms have little or no relationship to one another.

Several metrics exist for estimating the optimal number of clusters. Among these, Bayesian Information Criterion (BIC) [33] is one of the most reliable [34], [30]. BIC is computed as a byproduct of EM clustering. Nevertheless, the metric is also commonly used for estimating K in both K-means and hierarchical clustering [34], [30]. We use BIC as the basis for assigning a value to K in our approach.

Briefly, BIC is a measure for comparing statistical models with different parameterization methods, different numbers of components, or both [31]. When developing a statistical model to fit given data, one can improve the fit by adding additional parameters. This may however result in overfitting. To avoid overfitting, BIC penalizes model complexity so that it may be maximized for simpler parameterization methods and smaller numbers of components (clusters, in our case) [31]. The larger the BIC value is, the better the fit and consequently the better the selected number of clusters. Adapting BIC to our application context is the subject of one of our research questions; see RQ3 in Section 5.6.3.

2.4 Related Work

As noted earlier, this article is an extension of a previous conference paper [13]. In this section, we discuss and compare with several other strands of related work in the areas of term extraction, clustering, and NLP.

2.4.1 Term Extraction

We organize our review of term extraction into two parts: (1) general literature and tools, and (2) relevant research in the subject field of this article, i.e., Requirements Engineering.

General literature and tools. Term extraction has been studied in many domains and under numerous titles, including terminology identification, terminology mining, term recognition, term acquisition, keyword extraction, and keyphrase detection [7]. Term extraction approaches can be broadly classified into three categories [35]: *linguistic*, *statistical*, and *hybrid*.

Linguistic approaches aim at specifying patterns for detecting terms based on their linguistic properties, e.g., their POS tags. For example, Bourigault [36] describes a linguistic approach for extracting terms by eliminating certain grammatical patterns like pronouns and determiners, and then using regular expressions based on POS tags to extract certain combinations of NPs. Aubin and Hamon [37] use a combination of chunking and parsing to extract both simple and complex NPs.

Statistical approaches select terms based on statistical measures such as frequency and length. For example, Jones et al. [38] develop a statistical approach for identifying keywords by assigning ranks to word sequences in a document in such a way that frequently-occurring sequences which have many frequently-occurring words receive a high rank. In a similar vein, Matsuo and Ishizuka [39] use the co-occurrence frequency of words and of sequences of words for identifying keywords.

Hybrid approaches are combinations of linguistic and statistical ones. For example, Barker et al. [8] first employ text chunking for identifying the NPs in a given text, and subsequently filter out terms that are unlikely to be keywords based on frequency and length.

Our approach is a linguistic one. We do not use statistical measures because these measures primarily serve as filters. Speaking in terms of classification accuracy metrics, filtering improves precision, i.e., it decreases false positives. However, improvements in precision may come at the cost of losses in recall, i.e., increases in false negatives. For large and heterogeneous corpora, e.g., book and article collections, online commentary and – in the case of software – repositories of development artifacts, statistical measures, notably frequencies, provide a useful indicator for the importance of terms. Over such corpora, filtering based on statistical measures is often essential in order to achieve reasonable precision. In requirements documents, however, every individual statement is expected to have a clear and non-redundant purpose. Therefore, terms in requirements documents, regardless of their statistical characteristics such as frequencies, have the potential to bear important content. Consequently, using statistical filters over requirements documents is likely to have a significant negative impact on recall. In our work, we take a conservative approach toward filtering. In particular, we do not filter any terms on statistical grounds. The only filter we apply is a linguistic one over common nouns, as we explain in Section 3.

With regard to tool support for term extraction, several generic tools are already available, including the following:

- *JATE (Java Automatic Term Extraction toolkit)* [40] implements several term extraction techniques developed and used by the Information Retrieval (IR) community.
- *TextRank* [10] is a general text processing tool, with term extraction being one of its constituent parts. Extraction is performed based on POS tags, and an undirected graph in which edges represent pairwise relationships between terms based on their level of co-occurrence.
- *TOPIA* [9] is a widely-used Python library for term extraction based on POS tags and simple statistical measures, e.g., frequencies.
- *TermRaider* [11] is a term extraction module implemented as a plugin for the GATE NLP Workbench [41]. TermRaider uses advanced heuristics based on POS tags, lemmatization, and statistical measures.
- *TermoStat* [42] is a term extraction tool based on POS tags, regular expressions, and frequency-based measures.

All the aforementioned tools are based on hybrid techniques. As we demonstrate in Section 5, over requirements documents, our proposed term extraction technique yields better recall than these tools without compromising precision. Our work is further distinguished from these tools in that it clusters the extracted terms based on relatedness.

Term extraction in Requirements Engineering. Term extraction has been tackled previously in Requirements Engineering. Aguilera and Berry [43] and Goldin and Berry [44] present frequency-based methods for identifying terms that appear repeatedly in requirements. They refer to these terms as "abstractions" which are likely to convey important domain concepts. Popescu et al. [45] extract terms from restricted natural language requirements using parsing and parse relations. Zou et al. [46] use a POS tagger for extracting single- and double-word noun phrases, and then filter the results based on frequency measures and certain heuristics. Kof et al. [47] use POS tags, named-entity recognition, parsing, and heuristics based on sentence structures for extracting domain-specific requirements terms. Dwarakanath et al. [48] use parsing for extracting the phrases of requirements documents and then filter the results based on heuristics and frequency-based statistics. And, Ménard and Ratté [49] extract domain-specific concepts from business documents (including requirements) using POS tag patterns and various heuristics.

In addition to the above work which concentrates on requirements, there is work on term extraction from the broader set of software artifacts, aimed at improving software comprehension. For example, Sridhara et al. [50] extract terms from code and documentation, and subsequently use semantic similarity measures for assisting software developers in identifying information that is relevant to a search query. In a more specific context, Abebe and Tonella [51] and Abebe et al. [52] use NLP parsing for supporting programmers in the task of identifying maintenance-related terms from the names of methods, classes, and attributes in software code.

All these existing threads of work have helped us in better tailoring our term extraction technique to requirements. The main technical novelties contrasting our work from earlier work are the following: (1) We use text chunking for identifying candidate terms; text chunking is more accurate and

generalizable than pattern-based techniques based on POS tags, and more robust and scalable than parsing for phrase detection [53]. And, (2) we apply clustering for grouping candidate terms. Furthermore, and for reasons discussed earlier, our term extraction technique does not use statistical filters.

2.4.2 Clustering

Grouping together (clustering) related terms has been studied in the field of Information Retrieval. Existing approaches rely on pre-defined patterns of POS tags for identifying relatedness. Daille [54] uses the prefix POS tags of NPs for identifying adjectival and prepositional modifications. For example, their approach would group the terms "package" and its adjectivally-modified form "biodegradable package". Similarly, Bourigault and Jacquemin [55] group related terms based on patterns of noun modifiers. For example, their approach would group the terms "cylindrical cell" and "cylindrical bronchial cell". The main difference between our approach and the above is that, instead of patterns, we use syntactic and semantic similarity measures for detecting relevance. For unrestricted natural-language content, an exhaustive enumeration of all patterns of interest is very difficult; pattern-based approaches are therefore prone to incompleteness. Our approach does not suffer from this issue. Furthermore, our approach can systematically deal with morphological and semantic relatedness, which existing pattern-based approaches do not address sufficiently.

In the field of Requirements Engineering, clustering has been already applied for a variety of purposes. Ferrati et al. [56] cluster requirements statements in order to organize them into cohesive sections within requirements documents. Arafeen and Do [57] use requirements clustering as an enabler for test case prioritization. Chen at. al. [58] cluster related requirements for building product-line feature models. Duan et al. [59] apply and empirically evaluate the usefulness of different clustering techniques for grouping related development artifacts (requirements, test cases, classes, etc.) and supporting traceability. They further provide guidelines for selecting the number of clusters in this application context. Finally, Mahmoud [60] uses clustering for classifying non-functional requirements and tracing them to functional requirements.

Our application of clustering is guided by the same principles as in the above threads of work. Nevertheless, these threads do not use clustering to achieve the same end goal as ours, which is grouping together candidate glossary terms. A critical prerequisite for applying clustering effectively is to identify, for a specific analytical task, a suitable combination of a clustering algorithm and similarity measures. Doing so requires empirical studies that focus on the task at hand. To our knowledge, an empirical study similar to the one in this article does not exist for the task of building glossaries.

2.4.3 Natural Language Processing

In addition to term extraction, reviewed in Section 2.4.1, there are several other Requirements Engineering tasks in which NLP has been used for automation. These tasks include, among others, identification of inconsistencies and ambiguities [61], [62], [63], [64], [65], [66], [67], requirements tracing [59], [68], [69], [70], [71], [72], requirements change analysis [73], detection of redundancies and implicit requirements relations [74], [75], extraction of models from requirements [76], [77], identification of use cases [78], markup generation for legal requirements [79], [80], enforcement of requirements templates [16], synthesis of user opinions about features [81], [82], and requirements identification [83].

The NLP techniques we use in this article are not new to the Requirements Engineering community. Nevertheless, the NLP techniques underlying our approach, notably text chunking and semantic similarity measures, have not been systematically studied alongside clustering before. Furthermore, empirical studies that investigate the effectiveness of NLP over industrial requirements remain scarce. The case studies we report on in this article take a step toward addressing this gap.

3 Approach

Fig. 5 shows an overview of our approach. Given a (natural-language) requirements document, we first construct a list of candidate glossary terms. In the next step, we compute a similarity matrix for the extracted terms. In the third and final step, we cluster the terms based on their similarity. The rest of this section elaborates each of these steps.

3.1 Extracting Candidate Glossary Terms

Using text chunking, this step extracts a set of candidate glossary terms from a given requirements document. As explained in Section 2.1, from the results of text chunking, we need only the NPs. Following text chunking, all the extracted NPs are processed and cleared of determiners, pre-determiners, cardinal numbers, and possessive pronouns. For example, "the system operator" is reduced to "system operator". Furthermore, plural terms are transformed into singular terms using lemmatization. For example, "GSI anomalies" is transformed into "GSI anomaly".

Subsequently, we refine the list of terms by applying the heuristics listed in Table 1. The first heuristic in the table aims at re-establishing the context that may have been lost for some NPs. Specifically, text chunking decouples concepts from their attributes / subparts connected by "of" or a possessive 's. For example, the phrase "status of GSI component" gives rise to two NPs: "status" and "GSI component". However, "status" is unlikely to be useful as a term outside its context. To capture this intuition, we add to the list phrases of the form: *NP of NP* and *NP's NP*.

The second heuristic adds to the list of terms: (1) any all-capital token appearing within some NP, and (2) any continuous sequence of tokens marked as proper nouns (NNPs) by the POS tagger within the boundary of an individual NP. For example, the token "GSI" in "GSI component" will be added to the list as an independent term and so will "Ground Station Interface" if an NP such as "Ground Station Interface component" is already on the list. This is despite the fact that "GSI" and "Ground Station Interface" may never appear in the document as NPs. This heuristic is targeted at ensuring that potential abbreviations and named entities will have dedicated entries in the list of terms.

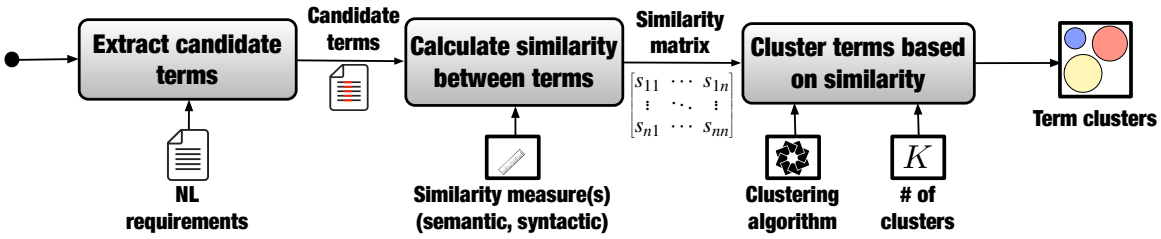


Fig. 5. Approach overview.

TABLE 1
Heuristics applied to the results of text chunking.

Heuristic	Description	Example
NP of NP / NP's NP	The combination of two NPs separated by "of" or a possessive 's is added to the list of terms.	"status of GSI component" *The second NP, i.e., "GSI Component" provides the context for the first NP, i.e., "status".*
Special tokens and sequences of proper nouns	Abbreviations and sequences of proper nouns (marked as NNP by the POS tagger) within individual NPs are added as independent entries to the list of terms.	"GSI component" / "Ground Station Interface component" *The abbreviation "GSI" and the sequence of proper nouns "Ground Station Interface" are extracted and added as independent entries to the list of terms.*
Common nouns	Single-word phrases that have a meaning in an English dictionary are filtered out.	"status" *This NP is unlikely to contribute to the glossary unless coming alongside its context, e.g., as in "status of GSI component", or is capitalized to signify a probable proper noun.*

The last heuristic in Table 1 is for filtering common nouns. By a common noun, we mean a single-word noun, e.g. "status", that is found in an (English) dictionary. We use the WordNet dictionary [84] for word lookup operations. The rationale for filtering common nouns is that these nouns are often generic and polysemous, and thus, outside their context, unlikely to contribute to the glossary [55]. Single-word nouns that are not found in a dictionary or are capitalized will be retained in the list of terms.

Finally, we remove any duplicates from the list of terms. Fig. 1(b) shows the list of terms derived from the requirements of Fig. 1(a) through the process described above.

Measuring the accuracy of our term extraction technique and comparing the accuracy to that of generic term extraction tools is the subject of one of our research questions; see RQ1 in Section 5.6.1.

3.2 Computing Similarities between Terms

This step computes a similarity matrix to capture the degree of relatedness between every pair of candidate terms extracted in the previous step. To compute this matrix, we consider three alternative strategies [73]:

1) *syntactic only*, where a similarity, $\mathscr{S}_{syn}(t,t')$, is computed for every pair (t,t') of terms using a *syntactic* measure, e.g., SoftTFIDF;
2) *semantic only*, where a similarity, $\mathscr{S}_{sem}(t,t')$, is computed for every pair (t,t') of terms using a *semantic* measure, e.g., JCN;
3) *combined syntactic and semantic*, where, given a syntactic measure *syn* and a semantic measure *sem*, we take, for every pair (t,t') of terms, $\max(\mathscr{S}_{syn}(t,t'),\mathscr{S}_{sem}(t,t'))$. Using max. is motivated by the complementary nature of syntactic and similarity measures [29], [85].

Choosing the best strategy from the above and the specific similarity measures to use are addressed by RQ2 and RQ4; see Sections 5.6.2 and 5.6.4.

3.3 Clustering Terms

In this step, we cluster the candidate terms based on their degree of relatedness. The inputs to this step are the similarity matrix (or dually, the distance matrix in the case of hierarchical clustering), the choice of clustering algorithm to use, and the number of clusters, K, to generate. As the result of clustering, the terms are grouped into K partitions. For example, Fig. 1 (c) shows a partitioning of the terms in Fig. 1 (b) with $K = 8$. The clustering algorithm applied here is EM and the similarity measure used is SoftTFIDF alone (i.e., without an accompanying semantic measure).

In our empirical evaluation of Section 5, we investigate all the key questions related to tuning clustering for use in our application context. Specifically, identifying the most accurate clustering algorithm(s) is addressed in RQ2 and RQ4. Choosing a suitable K is tackled in RQ3; see Section 5.6.3.

4 Tool Support

We implement our approach into a tool named REGICE (REquirements Glossary term Identification and ClustEring tool). The components of REGICE are shown in Fig. 7.

First, the requirements provided by the user are processed by a text chunker in the GATE NLP Workbench [41]. GATE is

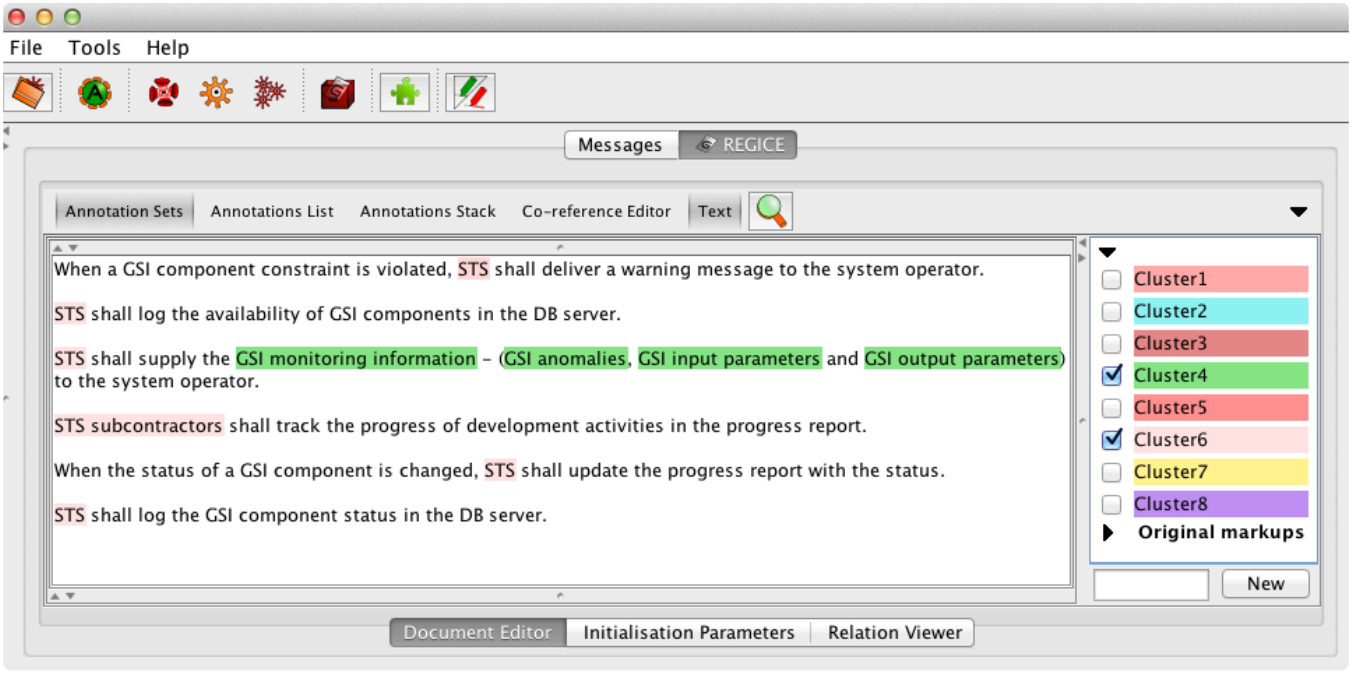


Fig. 6. Screenshot of REGICE (implemented in GATE [41]) with two computed clusters highlighted.

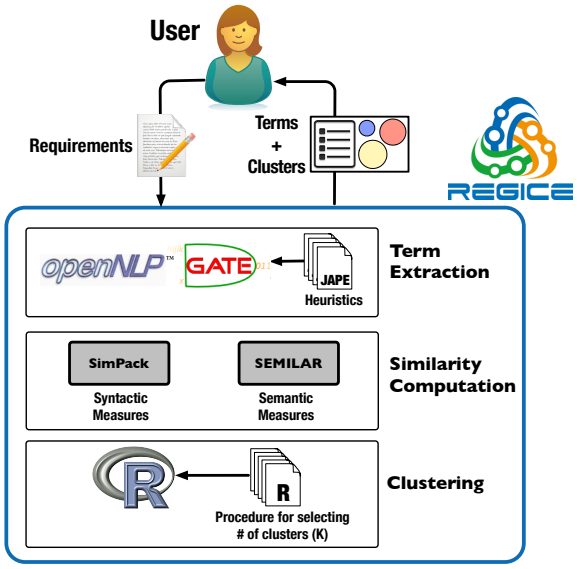


Fig. 7. Tool Overview.

an infrastructure built over a large collection of heterogeneous NLP technologies, making it possible for these technologies to interact and work together. Within GATE, there are several alternatives for implementing the text chunking pipeline discussed in Section 2.1. Among the alternatives, we use OpenNLP [86]. This choice is based on a comparative study in our previous work [16], where we found the OpenNLP chunking pipeline to be one of the most accurate and robust alternatives over requirements documents. The heuristics we apply for refining the results of text chunking (Section 3.1) are implemented using scripts written in GATE's regular expression language, JAPE (Java Annotation Patterns Engine).

We use SimPack [87] for computing syntactic similarities and SEMILAR [88] for computing semantic similarities between the extracted terms. Both libraries are Java-based. The default syntactic measure in REGICE is SoftTFIDF. No semantic measure is used by default, although the user has the option to choose any of the semantic measures provided by SEMILAR. Our default choices are based on our empirical observations (from RQ4) in Section 5.

For clustering and computing the BIC (Section 2.3.2), we use the R statistical toolkit [89]. The default clustering algorithm in REGICE is EM, again based on our empirical observations (from RQ4). The R library used for clustering depends on the choice of the clustering algorithm. K-means and all variants of hierarchical clustering are done using the `stats` package. EM clustering and BIC computation are done using the `mclust` package [31].

The number of clusters to generate is determined automatically using R scripts that implement the guidelines derived from our empirical results (RQ3). The user has the option to override the automatic recommendation for the number of clusters and provide a different number of clusters.

Once run on a given set of requirements, REGICE yields two outputs: (1) a flat list of extracted terms written out to a file, and (2) a set of term clusters. For presenting the clusters, the tool can either store them in a file as labeled sets, similar to what is shown in Fig. 1(c), or visually highlight the clusters over the requirements through GATE's user interface. We illustrate this user interface in the screenshot of Fig. 6. The requirements in this screenshot are from the example of Fig. 1. As shown by the right panel of the screenshot, each cluster is represented as an annotation type. When a cluster (annotation type) is selected, all the terms in that cluster are highlighted in the document. This visual representation has the advantage that it preserves the context where each term in a given cluster appears.

When working with clusters, we expect that the user would focus on one cluster at any given time. Once a cluster has been selected for inspection, the user may first want to ensure that there is no unknown and potentially undesirable synonymy in the cluster. For instance, in the example of Fig. 1(c), the user may elect to eliminate from the requirements document either "GSI component status" or "status of GSI component" in order to make the terminology consistent. In the next step, the user may select from the cluster under investigation the terms that need to be defined in the glossary, and subsequently proceed to write definitions for the selected terms. While a cluster may not directly contribute to writing definitions for the terms in it, our empirical results (Section 5.6) suggest that clusters do provide useful assistance in this task by bringing together the related terms, e.g., domain concepts and their attributes. Finally, the user may choose from a cluster any related terms that need to be mentioned alongside a glossary term.

The components of REGICE have been integrated together via glue code written in Java. REGICE, including the R and JAPE scripts, is approximately 2000 lines of code excluding comments and third-party libraries. The tool is available at:

https://sites.google.com/site/svvregice/

5 Empirical Evaluation

We evaluate our term extraction and clustering techniques over three industrial case studies. In this section, we elaborate the research questions that motivate our evaluation and report on the design, execution and results of the case studies.

5.1 Research Questions

Our evaluation aims to answer the following research questions (RQs):

RQ1. How accurate is our approach at extracting glossary terms? A set of candidate terms is accurate if it neither includes too many unwanted terms (false positives) nor misses too many desired terms (false negatives). The aim of RQ1 is to evaluate the accuracy of text chunking, enhanced with our heuristics, at detecting glossary terms.

RQ2. Which similarity measure(s) and clustering algorithms(s) yield the most accurate clusters? The choice of similarity measures and clustering algorithm can have a major impact on the quality of the generated clusters. The aim of RQ2 is to examine alternative combinations of similarity measures and clustering algorithms, and identify the best alternatives in terms of accuracy.

RQ3. How can one specify the number of clusters? A bad choice for the number of clusters can compromise the accuracy of clustering and potentially render the resulting clusters useless. The aim of RQ3 is to develop systematic guidelines for choosing an appropriate number of clusters for a specific requirements document.

RQ4. Which of the alternatives identified in RQ2 are the most accurate when used with the guidelines from RQ3? RQ2 uses an averaging metric for identifying the most accurate clustering algorithms and similarity measures. This metric is *not* a direct indication of accuracy at a fixed number of clusters. From a practical standpoint, one needs to know which combinations of clustering algorithms and similarity measures are best when the number of clusters is set as per the recommendation from RQ3. The aim of RQ4 is to find the combinations that work best with the guidelines of RQ3.

RQ5. Does our approach run in practical time? One should be able to perform candidate term extraction and clustering reasonably quickly, even when faced with a large number of requirements. The aim of RQ5 is to investigate whether our approach has a practical running time.

RQ6. How effective is our clustering technique at grouping related terms? Clustering can be a useful assistance to analysts during glossary construction only if the generated clusters are sufficiently accurate. Drawing on the clustering accuracy results from our case studies, RQ6 argues about the overall effectiveness of our clustering technique.

RQ7. Do practitioners find the clusters generated by our approach useful? Ultimately, our clustering technique is valuable only if practitioners faced with real Requirements Engineering tasks find the generated clusters useful. RQ7 is aimed at assessing the perceptions of the experts involved in our case studies about the usefulness of the generated clusters.

5.2 Description of Case Studies

Table 8 provides, for each of our case studies, a short description, the case study domain, and the number of requirements statements in the respective requirements document.

The first case study, hereafter *Case-A*, concerns a software component developed by SES Techcom – a satellite communication company – for a satellite ground station. Case-A has 380 requirements. The second case study, hereafter *Case-B*, concerns a safety evidence management system built in an EU project, OPENCOSS (http://www.opencoss-project.eu), with participation from 11 companies and 4 research institutes. Case-B has 110 requirements. The third case study, hereafter *Case-C*, concerns a satellite data dissemination network developed in a European Space Agency (ESA) project with participation from several telecommunication companies. Case-C has 138 requirements. Case-A and Case-C are proprietary; whereas Case-B is public. To facilitate replication, we make the material for Case-B available on our tool's website (see Section 4).

For each case study, we involve a subject matter expert with in-depth knowledge about the respective case. In Case-A and Case-B, the experts were requirements analysts who were closely involved in drafting the requirements; and in Case-C, the expert was the project manager.

We have used the requirement documents in Case-A and Case-B as case study material before [16]. In both cases, the requirements writers had made a conscious attempt to structure the requirements sentences according to Rupp's template [4]. Specifically, 64% of the requirements in Case-A and 89% of the requirements in Case-B conform to Rupp's template [16]. We are using Case-C as case study material for the first time. No particular template was used in the requirements of Case-C.

Case	Description	Domain	Number of Requirements
Case-A	Requirements for a software component in a satellite ground station	Satellites	380
Case-B*	Requirements for a safety evidence management system	Safety certification of embedded systems	110
Case-C	Requirements from a data dissemination network solution for satellites.	Satellites	138

* *The material for Case-B is available on our tool's website (see Section 4).*

Fig. 8. Description of case studies.

In Section 6, we argue why the use of a template in Case-A and Case-B does not pose major validity threats. Except for the elicitation of glossary terms for Case-A and Case-B (see Section 5.4.1), all the empirical work reported in this section was conducted as part of our current research.

5.3 Case Selection Criteria

We had the following criteria in mind when selecting our case studies:

- We were interested in requirements documents that are reasonably large (> 100 requirements), first, because automated term extraction and clustering is unlikely to provide compelling benefits over very small requirements documents, and second, because we would not be able to adequately evaluate the execution time of our approach (RQ5) using small documents.
- We wanted to cover cases from different domains. In general, conducting multiple case studies is useful for mitigating external validity threats. In our investigation, increasing external validity is particularly important for RQ3, due to the impact that the choice of the number of clusters has on the quality of the generated clusters.
- We wanted to work on cases where we could have direct access to subject matter experts. Particularly, to evaluate the accuracy of our approach, we need a gold standard, covering both the ideal glossary terms and the ideal clusters of related terms. Building a trustworthy gold standard requires deep knowledge about the problem domain and a significant level of commitment. Consequently, ensuring the availability of experts throughout our investigation was an important criteria.
- We were interested in requirements from recent or ongoing projects. Old requirements are unsuitable for our evaluation, both due to the experts' potential lack of interest to revisit these requirements, and also due to the high likelihood that the experts would not be able to readily remember all the details.

The cases we have selected satisfy the above criteria.

5.4 Data Collection Procedure

Data collection was targeted at building the ideal set of glossary terms and clusters. We elicited the ideal glossary terms directly from the experts. As for the ideal clusters, they were elicited indirectly and through the construction of a domain model. Below, we detail the process for glossary term elicitation and domain model construction. The process for deriving ideal clusters from a domain model is discussed as part of our analysis procedure (see Section 5.5.1). Note that the domain model and ideal clusters are *only for evaluation purposes* and not a prerequisite for applying our approach.

5.4.1 Glossary Term Elicitation

Despite the requirements in all our case studies having reached stability, no glossary was available for the requirements yet. To identify the glossary terms, we held walkthrough sessions with the respective expert in each case study. In these sessions, the expert would first read an individual requirements statement and then identify the glossary terms in that particular statement. The expert was asked to specify all the glossary terms in a given statement, irrespective of whether the terms had been already seen in the previous statements. When the expert was doubtful as to whether a term belonged to the glossary, they were instructed to include the term rather than leave it out, as recommended by glossary construction best practices [4].

The researchers' role in the walkthrough sessions was limited to moderating the sessions and keeping track of the experts' choices about the glossary terms. Once the expert in each case study reviewed all the requirements statements in the case study, a duplicate-free list of the terms chosen by the expert for the glossary was created. For Case-A and Case-B, these lists were built as part of our previous work [16]. The experts were allowed to revise these lists during domain model construction (described next), which took place after glossary term elicitation. The final lists of terms are used as the gold standard for answering RQ1.

5.4.2 Domain Model Construction

To evaluate the accuracy of our clustering technique, we need a set of ideal clusters. Rather than eliciting the ideal clusters directly, we first build a domain model – a conceptual representation of a domain – and then *infer* the ideal clusters from this domain model using the procedure described later. Intuitively, we would like each ideal cluster to bring together some glossary term and its "related" terms. The role of a domain model in this context is to specify what "related" means for every glossary term.

We observe that behind every requirements document, there is a domain model. This domain model may never be built

explicitly, or may be partial when it is built. Nevertheless, the observation has useful implications in terms of evaluating our approach. Particularly, given a domain model and a mapping from each concept and attribute of this model onto the terms in the requirements document, one can come up with a systematic procedure for inferring the ideal clusters (Section 5.5.1). Such a procedure presents two key advantages: First, it alleviates the need for the domain experts to construct the ideal clusters manually – a task that is very laborious for large requirements documents such as those in our case studies. Second, although one can never entirely remove subjectivity from how a domain model is constructed and how relatedness is defined, by building an explicit domain model and formulating relatedness in a precise way, one can subject our evaluation process to scientific experimentation.

For the purposes of our evaluation, the main property we seek in a domain model is the following: Given a glossary term t, the domain model should be able to give us all the terms in the underlying requirements document that are conceptually related to t. We limit conceptual relationships to *specializations*, *aggregations* and *compositions*. Specializations represent is-a relationships. Aggregations and compositions both denote containment relationships, with the difference being that, in aggregations, the contained objects can exist independently of the container; whereas in compositions, the contained objects are owned by the container and thus cannot exist independently of it. Specializations, aggregations, and compositions constitute some of the most basic relationships between concepts and are thus instrumental for capturing relatedness between terms.

In Fig. 9, we show a small (and sanitized) fragment of the domain model for Case-A. We use UML class diagrams for expressing domain models, as is common in object-oriented analysis [90]. In the figure, compositions are shown using a solid diamond shape and aggregations – using a hollow diamond shape.

We note that a domain model is not merely a structured representation of the content of a requirements document. This model further has to account for the tacit information that is not reflected in the requirements but is yet essential for properly relating the terms in the requirements. Examples of such tacit information in the class diagram of Fig. 9 are the composition associations from GSI to GSI Monitoring Information and GSI Component.

We associate each element (concept or attribute) x in the domain model with a set, $Var(x)$, of variant terms that are conceptually equivalent to x. For example, consider the availability attribute of GSI Component in the model fragment of Fig. 9. This attribute is referred to in the requirements document using three variant terms: "availability" (where the link to GSI Component is implicit), "availability of GSI Component" and "GSI component's availability". To avoid clutter in the figure, we have not shown $Var(x)$ when this set has only one term and that term coincides with the name label of x.

We use the notion of *Var* to describe how we built the domain models. We first reviewed the requirements document in each case study to identify all variants of the glossary terms elicited previously from the respective expert (Section 5.4.1).

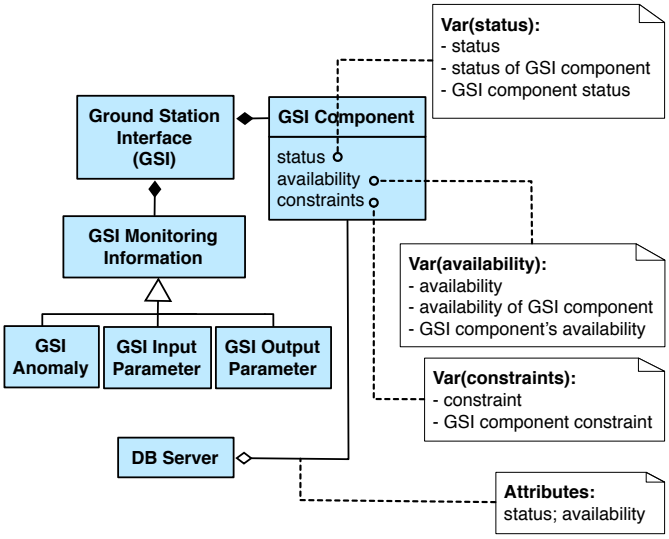


Fig. 9. A fragment of the domain model for Case-A.

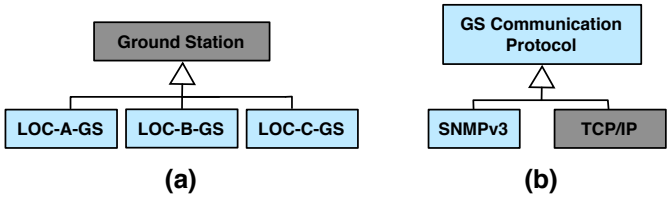


Fig. 10. Domain model versus glossary: grayed-out model elements have no corresponding glossary terms.

Let T be the set of glossary terms and all variants thereof, discovered through the aforementioned review. We built our domain model $\mathscr{M}$ for each case study in a way to ensure that all the terms in T were represented by some element x in $\mathscr{M}$, i.e., to ensure that $T \subseteq \bigcup_{x \in \mathscr{M}} Var(x)$. For our purposes, we would have liked $\mathscr{M}$ to represent nothing but the terms in T, i.e. to have $T = \bigcup_{x \in \mathscr{M}} Var(x)$. However, we found this constraint to be restrictive in that it could reduce the logical completeness of the domain model. We illustrate this point using the domain model fragments (from Case-A) that are shown in Fig. 10.

In the model fragments of Fig. 10, the grayed-out elements, Ground Station and TCP/IP, have no corresponding terms in the glossary although both elements have corresponding terms in the requirements document. In the model fragment of Fig. 10(a), the expert decided that the specific ground stations built at locations A, B, and C (actual locations sanitized) would need to be defined in the glossary; whereas, the abstract concept of ground station would not. A similar situation applies to the model fragment of Fig. 10(b): although it is important, for completeness reasons, to model TCP/IP as a protocol alongside SNMPv3, the expert did not see a need to define TCP/IP in the glossary because it is a widely-known and standard protocol.

In general, we attempted to closely orient the domain models in our case studies around the glossary terms. Nevertheless, in situations like those illustrated in Fig. 10, we opted to keep the non-glossary-related elements in the domain model for

completeness. The domain models were built collaboratively with the involved experts. As noted in Section 5.4.1, the experts were allowed to refine their choice of glossary terms based on insights gained during domain model construction.

Finally, to be able to properly handle requirements about data storage and transfer, we made a modeling decision that we illustrate using requirements R2 and R6 of Fig. 1(a). These requirements envisage that "DB server" shall store the status and availability of "GSI Component". As shown in Fig. 9, we model the relationship between DB Server and GSI Component as an aggregation, while keeping track of any specifically-named attributes, here, status and availability, that participate in the relationship. For deriving ideal clusters from such an aggregation, as we explain in Section 5.5.1, we use the participating attributes rather than the contained concept itself. This strategy helps make the ideal clusters more precise and better aligned with the requirements document.

5.4.3 Expert Interview Survey

We conducted an interview survey with the experts in our case studies in order to assess the experts' perceptions about the usefulness of our approach. Specifically, we chose a subset of the generated clusters in each case study, and asked the expert in that case study to evaluate these clusters individually on the basis of the three statements shown in Fig. 11. For Case-A and Case-C, the clusters in the survey are a random selection of 20 from our tool's output when executed with the default settings presented in Section 4. For Case-B, the tool yields 27 clusters, all of which are covered in the survey.

The statements in Fig. 11 address three important tasks that analysts need to perform during the construction of a glossary: Statement 1 concerns the identification of related terms. Statement 2 concerns writing definitions for the glossary terms. The rationale for including Statement 2 in our survey is that the additional context provided by a cluster (when compared to disparate individual terms) can help the analysts in writing more precise definitions for the glossary terms.

Statement 3 addresses a specific type of related terms, namely variations (synonyms). Although related terms are already covered by Statement 1, we elected to have a dedicated statement about variations, since variations can potentially be undesirable: from our experience, we observe that industrial requirements are prone to containing *unintended* variations, both due to the flexibility of natural language and also due to differences in terminology and style across different individuals and organizations. It is important to bring such variations to the attention of the analysts, so that they can take appropriate action. Statement 3 specifically examines whether the generated clusters are good means for identifying variations, which in many cases are unintended and potentially unknown.

The survey for each case study was conducted in a single session. To avoid interviewee fatigue, we limited the sessions to a maximum duration of 1 hour each. At the beginning of a session, we introduced to the (respective) expert the statements in Fig. 11 along with examples clarifying the motivation behind each statement. The relationship between Statement 1 and Statement 3 was further highlighted to the expert.

In the next step, the expert was asked to review the selected clusters in succession, and for each cluster, express their opinion about Statements 1, 2, and 3 on a five-point Likert scale [91]. For Statement 3, since not all clusters necessarily contain variations, the expert had an additional choice, "Not Relevant", to be used when they believed that a certain cluster did not contain any variations. The expert was reminded that, for all three statements, the opinion should be based on the glossary terms they saw in the cluster being reviewed. In particular, the expert was told that, if they did not see any glossary terms in a cluster, they should refrain from choosing either "Strongly Agree" or "Agree", although they may still see some benefit in the information provided by the cluster.

To ensure that the experts had a correct understanding of the statements in Fig. 11, we asked each expert to verbalize their reasoning for the first five clusters that they reviewed.

We note that we conduct one interview per case study. Ideally, one should have interviews with multiple experts in each case to enable comparison and increase the reliability of the results. In our work, having more than one interview for each case study was infeasible because any respondent would have to have full familiarity with the requirements before they could meaningfully answer the interview questions. To mitigate the effect of potential expert errors, our interview covers a reasonably large number of clusters (at least 20 clusters, as discussed earlier) for each case study.

5.5 Analysis Procedure

5.5.1 Inferring Ideal Clusters

Equipped with a domain model $\mathscr{M}$ and a function $Var(x)$ for every concept and attribute $x \in \mathscr{M}$ (as defined in Section 5.4), we infer the ideal clusters as we describe next.

Ideal clusters are created around concepts, with the concept attributes contributing to some of the clusters. For every concept $c \in \mathscr{M}$, we add to the set of ideal clusters one cluster, I, computed as follows: Let $a_1, \ldots, a_k$ denote c's attributes, and let $c_1, \ldots, c_n$ be the set of (parent) concepts that are directly or indirectly specialized by c (via specialization).

$$
\begin{aligned}
I = {} & Var(c) \cup \\
& \textstyle\bigcup_{1 \le i \le k} Var(a_i) \cup \\
& \textstyle\bigcup_{1 \le i \le n} Var(c_i) \cup \\
& \textstyle\bigcup \{ Var(a) \mid a \text{ is an attribute of some } c_i;\ 1 \le i \le n \} \cup \\
& \textstyle\bigcup \{ Var(s) \mid s \text{ is a sibling of } c \text{ via some } c_i;\ 1 \le i \le n \}.
\end{aligned}
$$

For example, let c be the GSI Anomaly concept in Fig. 9. We create a cluster by grouping together the following: c's variant terms (only, "GSI Anomaly"); variant terms for c's attributes (none); variant terms for c's parents ("GSI Monitoring Information") and parents' attributes (none); and variant terms for c's siblings ("GSI Input Parameter" and "GSI Output Parameter").

The above process captures relatedness between each concept and its attributes as well as between each concept and other concepts that immediately relate to it via the domain model's inheritance hierarchy. To deal with compositions and aggregations, we follow a separate process: let c_1 and c_2 denote the two ends of a

Statement 1. I find this cluster helpful for identifying the related terms for a glossary term.
❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree

Statement 2. As the result of seeing this cluster, I can define a glossary term more precisely than I originally had in mind.
❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree

Statement 3. I find this cluster helpful for identifying the variations (synonyms) of a glossary term.
❑ Strongly Agree ❑ Agree ❑ Neutral ❑ Disagree ❑ Strongly Disagree ❑ Not Relevant

Fig. 11. Statements for assessing the usefulness of a cluster.

composition or aggregation association. We add one cluster $J = Var(c_1) \cup Var(c_2)$ to the set of ideal clusters for each such association. For example, consider the two composition associations in Fig. 9. These induce the following clusters: {"Ground Station Interface","GSI Monitoring Information"} and {"Ground Station Interface","GSI Component"}.

The only exception to the above are aggregations in which explicitly-named attributes of the contained concept participate (see Section 5.4.2). For such aggregations, we bypass the contained concept and use the named attributes directly for the derivation of ideal clusters. For example, for the aggregation between DB Server and GSI Component in Fig. 9, we create one cluster for each status and availability. This yields two ideal clusters: (1) $Var(\text{DB Server}) \cup Var(\text{status})$, and (2) $Var(\text{DB Server}) \cup Var(\text{availability})$.

Our treatment of composition and aggregation associations is motivated by the fact that while a container concept (e.g., Ground Station Interface) is related to each of the contained concepts, there is a weaker or no relationship between the contained concepts (e.g., GSI Component and GSI Monitoring Information). Hence, putting the contained concepts together into the same cluster, only because they happen to be contained by the same container concept, does not seem reasonable.

After creating the ideal clusters in the manner described above, we remove duplicates and any ideal cluster I that is properly contained in some other ideal cluster I' (i.e., if $I \subset I'$). We use the resulting clusters as the gold standard for evaluation. The ovals in Fig. 18 show the ideal clusters for the example of Fig. 1. We discuss Fig. 18 further when addressing RQ6 (Section 5.6.6).

5.5.2 Evaluation Procedure

There are two main evaluation procedures underlying our empirical results: one is for assessing the accuracy of candidate terms, and the other – for assessing the accuracy of clusters:

Accuracy of candidate terms. We use standard classification accuracy metrics, *precision* and *recall* [19], to evaluate the accuracy of candidate terms (step 1 of the approach in Fig. 5). The task at hand is to determine which extracted terms belong to the glossary and which ones do not. The extracted terms that belong to the glossary are True Positives (TP), and the ones that do not belong are False Positives (FP). The glossary terms that are not extracted by our tool are False Negatives (FN). Precision accounts for the quality of results, i.e., smaller number of FPs, and is computed as TP / (TP+ FP). Recall accounts for the coverage, i.e., smaller number of FNs, and is computed as TP / (TP + FN). We use F-measure [19] to combine precision and recall into one metric. F-measure is computed as: 2 × Precision × Recall / (Precision + Recall).

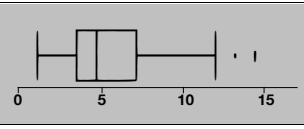
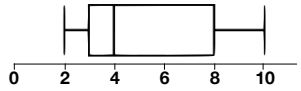
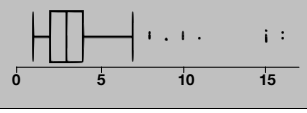
Accuracy of clustering. The choice of metrics for evaluating the accuracy of clustering (step 3 of the approach in Fig. 5) is not as straightforward. A wide range of metrics exist to this end, each with its own advantages and limitations. Clusters are typically evaluated across two dimensions: homogeneity and completeness [92]. Homogeneity captures the intuition that the data points (in our case, candidate terms) in a generated cluster should be originating from a single class (i.e., a single ideal cluster). Completeness captures the intuition that all the data points in a class (i.e., an ideal cluster) should be grouped together in one generated cluster. A common limitation of several clustering evaluation metrics, e.g., information-theoretic measures such as entropy, is that they are not particularly suited for situations where the clusters are overlapping [93].

In our work, meaningful handling of overlaps is essential: while our clustering approach (Section 3.3) produces partitions, i.e., non-overlapping clusters, our ideal clusters (Section 5.5.1) are overlapping. To evaluate the accuracy of clustering, we use a simple set of accuracy metrics – a standard generalization of precision, recall and F-measure for clusters [94] – which is straightforward to interpret in the presence of overlaps. Below, we outline the procedure for calculating these accuracy metrics for clusters. We discuss the implications of using partitioning clustering in RQ5; see Section 5.6.5.

Let $I_1, \ldots, I_t$ denote the set of *ideal* clusters, and let $G_1, \ldots, G_u$ denote the set of *generated* clusters.

- For every pair (I_i, G_j) $1 \leq i \leq t$; $1 \leq j \leq u$:
  - Let n_{ij} be the number of common data points between I_i and G_j.
  - $Precision(I_i, G_j) = \frac{n_{ij}}{|G_j|}$.
  - $Recall(I_i, G_j) = \frac{n_{ij}}{|I_i|}$.
  - $F\text{-measure}(I_i, G_j) = \frac{2 \times Precision(I_i, G_j) \times Recall(I_i, G_j)}{Precision(I_i, G_j) + Recall(I_i, G_j)}$.
- For every ideal cluster I_i $1 \leq i \leq t$:
  - Let G_r be the best match for I_i among generated clusters, i.e., $F\text{-measure}(I_i, G_r) \geq F\text{-measure}(I_i, G_j)$ for any $1 \leq j \leq u$. Let $P_{\text{best\_match}}(i)$ denote $Precision(I_i, G_r)$ and let $R_{\text{best\_match}}(i)$ denote $Recall(I_i, G_r)$.
- Let $n = \sum_{i=1}^{t} |I_i|$.
- Compute overall precision and recall as weighted-averages of the precisions and recalls of the ideal clusters:
  - $Precision = \sum_{i=1}^{t} \frac{|I_i|}{n} \times P_{\text{best\_match}}(i)$.
  - $Recall = \sum_{i=1}^{t} \frac{|I_i|}{n} \times R_{\text{best\_match}}(i)$.

TABLE 2
Information about the case studies.

Case	No. of glossary terms	No. of elements in domain model		No. of ideal clusters	Size distribution of ideal clusters	No. of generated clusters covered in interview survey
Case-A	140	Concepts (Classes)	238	119		20
		Attributes *	37			
		Specializations	143			
		Associations	58			
Case-B	51	Concepts (Classes)	35	29		27
		Attributes *	14			
		Specializations	5			
		Associations	29			
Case-C	200	Concepts (Classes)	274	142		20
		Attributes *	35			
		Specializations	110			
		Associations	123			

** The number of domain model attributes in all three case studies is proportionally small. The reason is that, in line with best practices, when there was uncertainty as to whether an element should be a concept (class) or an attribute, we modeled it as a concept.*

Overall precision and recall as defined above provide metrics for measuring the homogeneity and completeness of clusters, respectively. F-measure for clustering is computed as the harmonic mean of (overall) precision and recall. This generalization of accuracy metrics for clusters is used in Section 5.6 for answering RQ2, RQ4 and RQ6.

We note that our accuracy metrics for clusters are based on establishing a mapping from the ideal clusters to the generated ones, and not vice versa. This mapping is not necessarily surjective, meaning that not all the generated clusters may contribute to the calculation of the clustering accuracy measures. Our choice of metrics reflects the goal that we would ideally like to achieve via clustering, which is to group related terms so that an analyst can inspect the terms together rather than in isolation. In line with this goal, our metrics attempt to capture, for each ideal cluster, how good a match we can find in the generated clusters. The fact that our metrics may not account for all generated clusters does not pose a threat to the conclusions we draw based on the metrics. This is because: (1) the generated clusters are non-overlapping, and (2) any generated cluster unaccounted for by the metrics would still be composed of candidate terms and thus needs to be inspected by an analyst.

5.6 Results and Discussion

In this section, we describe the results of our case studies and answer the RQs stated in Section 5.1.

Table 2 provides overall statistics about the outcomes of data collection, showing, for each case study, the number of elicited glossary terms, the number of elements in the developed domain model, the number and size distribution of ideal clusters, and the number of clusters reviewed by the respective expert in the interview survey. For Case-A, a domain model existed beforehand. The researchers elaborated this model to achieve the desired characteristics detailed in Section 5.4.2. In Case-B and Case-C, no domain model existed a priori. The researchers built a domain model in each of these two cases by following standard practices for domain modeling [90], and in a way as to ensure the desired characteristics. In all three case studies, the resulting domain model was thoroughly validated with the involved expert.

5.6.1 RQ1. How accurate is our approach at extracting glossary terms?

Our terms extraction tool, REGICE, yielded 604 candidate terms for Case-A, 91 terms for Case-B, and 630 terms for Case-C. Fig. 12 shows the classification accuracy results for our term extraction technique and compares them against the results from five existing term extraction tools, outlined in Section 2.4.1. We note that one of these tools, JATE, implements several alternative term extraction techniques. The results in the chart of Fig. 12 are for the technique by Frantzi et al. [6], which, among the alternatives in JATE, has the best accuracy over our case studies.

As the chart indicates, our term extraction technique has better recall than the existing tools considered. Furthermore, and in terms of precision, our technique yields better results than all but TextRank in Case-A and Case-C. In both of these cases, the precision loss compared to TextRank is small (0.9% in Case-A and 1.2% in Case-C); whereas the gain in recall is large (23.6% in Case-A and 29.5% in Case-C). In other words, when compared to TextRank, our approach produces a small number of additional unwanted terms (false positives) in Case-A and Case-C; but, at the same time, our approach identifies a large number of desirable terms that TextRank misses.

For glossary term extraction, recall is a more important factor than precision. A low recall (i.e., a high number of false negatives) means that the analysts will miss many of the terms that should be included in the glossary. Low precision is comparatively easier to address, as it entails only the removal of undesired items (false positives) from a list of extracted

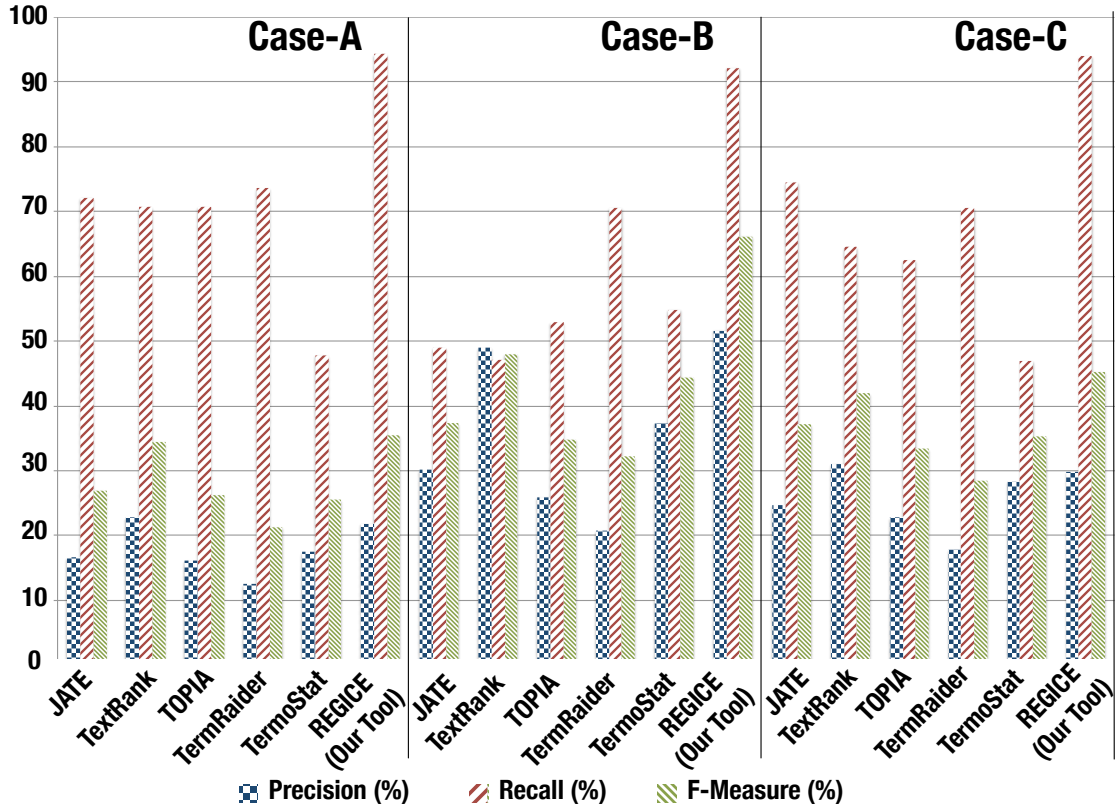


Fig. 12. Accuracy of terms extraction.

candidate terms. Given that our term extraction technique offers recall improvements of 20% or more over existing state-of-the-art tools, and at the same time, maintains or improves precision, it is reasonable to conclude that our technique is advantageous for the task of extracting glossary terms from requirements documents.

To determine whether the above observations have statistical significance, we use the z-score test for population proportions [95]. Here, the proportions of interest are: (1) true positives over the total number of terms extracted and (2) true positives over the total number of terms in the gold standard. The former proportion captures precision and the latter captures recall. Since there are five tools to compare with, as shown in Fig. 12, and three case studies, we conducted a total of 15 (i.e., 5×3) z-score tests for each of the two proportions. All tests are two-tailed with α=0.05. Based on these tests, we conclude that our tool yields significantly better precision than two of the tools compared with, namely TOPIA and TermRaider. The precision gains brought about by our tool (if any) are at best negligible when we compare with the remaining three tools. In contrast, the tests indicate that our tool yields significantly better recall than *all* the five tools compared with.

Our technique yields 8 false negatives in Case-A, 4 false negatives in Case-B, and 12 false negatives in Case-C. Of these 24 false negatives in total, 13 are explained by the heuristic we apply for filtering single-word common nouns (Table 1). From the remaining 11 false negatives, 4 are explained by our decision not to include VPs as candidate terms. The other 7 are due to limitations in the underlying NLP technology (i.e., mistakes made by the text chunking pipeline). Including single-word common nouns and VPs would address 17 out of the 24 false negatives. However, doing so would have a substantial and non-negligible negative impact on precision by introducing many additional false positives (precisely, 773 new false positives across the three case studies).

We further observe that VPs account for only 0.01% of the glossary terms in our case studies. This is consistent with the findings of Justeson and Katz [15] and their conclusion that VPs contribute $\leq 1\%$ of the terms in technical glossaries.

The results in the chart of Fig. 12 prompted an investigation as to why precision for Case-A and Case-C is lower across all techniques, including ours. To identify the cause, we asked the experts in Case-A and Case-C to explain the rationale in choosing the ideal glossary terms. We determined that their choices exclusively reflected the terms for the glossary of the specific requirements documents being analyzed. In other words, terms that were deemed common knowledge or were known to be already defined in the glossaries of related documents were excluded. In general, since such contextual factors and working assumptions are often tacit and thus unavailable to an automated tool, large variations may be seen in terms of precision across different projects. Nevertheless, recall, which is the primary factor for glossary term extraction as we argued above, will not be affected.

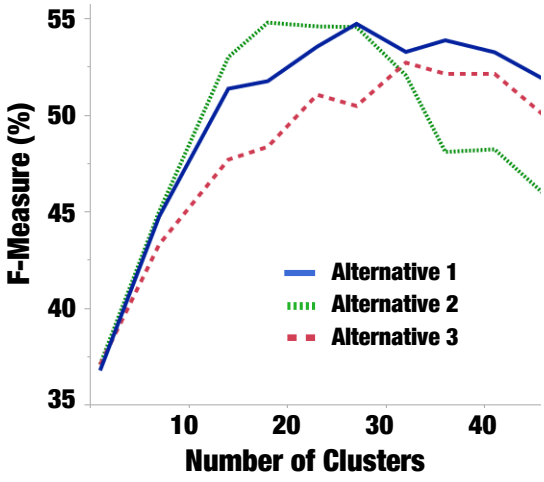


Fig. 13. F-measure curves for three different cluster computation alternatives.

5.6.2 RQ2. Which similarity measure(s) and clustering algorithms(s) yield the most accurate clusters?

We assess the accuracy of clustering based on the F-measure metric for clusters, defined in Section 5.5.2. As suggested by the discussion in RQ1, the set of candidate terms in a requirements document can be wider than the ones that are of interest for glossary construction. However, the generated clusters cover *all* candidate terms, not only those that are relevant to the glossary. To determine which similarity measure(s) and clustering algorithm(s) produce the best results relevant to the glossary, we need to discard in our analysis of accuracy the terms that are not relevant. Specifically, for the purpose of F-measure calculation, we prune from the generated clusters any term that is not in at least one of the ideal clusters.

We note that outside an evaluation setting, one cannot distinguish terms that are relevant to the glossary from those that are not. To preserve the realistic behavior of clustering, it is thus paramount to compute the generated clusters for all the candidate terms first and then prune the results for evaluation, as opposed to narrowing the set of candidate terms to those that are relevant and then clustering only the relevant terms.

To answer RQ2, we considered pairwise combinations of the 12 syntactic and 8 semantic similarity measures introduced in Section 2.2. Three semantic measures, *HSO, LESK, and LCH*, were discarded during initial analysis due to scalability issues. The total number of remaining combinations is $(12+1)\times(5+1)-1=77$. These combinations include configurations where an individual syntactic or semantic measure is applied on its own.

For clustering, we considered *K-means*, *Hierarchical* and *EM*, as introduced in Section 2.3.1. We experimented with 8 variant cluster distance functions (Table A.3 in the appendix) for hierarchical clustering. This brings the total number of clustering algorithms to 10. We evaluated each clustering algorithm in conjunction with all the 77 possible combinations of similarity measures, i.e., a total of $10\times 77=770$ alternative cluster computations per case study.

For a given cluster computation alternative, i.e., combination of a clustering algorithm and similarity measures, plotting F-measure against the number of clusters results in curves similar to those shown in Fig. 13.

The shape of these curves is explained as follows: When the selected number of clusters, K, on the x-axis is small, the size of the generated clusters is large, since the average size of clusters is inversely related to K. These large-sized clusters yield high recall but low precision, with an overall low F-measure. As K increases, large clusters become smaller and more cohesive. This brings about major increases in precision without a significant negative impact on recall, thus increasing F-measure. The F-measure peaks at some K value. This is where the generated clusters are closest to the ideal ones. Beyond this optimal K (the estimation of which is the subject of RQ3), increases in K will reduce F-measure, as recall begins to drop rapidly and losses in recall are no longer offset but gains in precision.

For a set of candidate glossary terms with a cardinality of n, increasing the value of K beyond $n/2$ would have little practical meaning, as the average size of clusters would fall below 2 terms per cluster. Generating clusters that are this small would defeat the purpose of clustering. To use curves similar to those in Fig. 13 as an evaluation instrument, we therefore restrict the upper range for K to a maximum of $n/2$ (upper bound).

For each of the 770 alternatives in a given case study, we plot an F-measure curve for 10 points on the x-axis at regular intervals, ranging from 1 to $n/2$. Note that n is the number of candidate terms in the case study in question. The reason we limit ourselves to 10 observation points is that a thorough analysis for all possible numbers of clusters is extremely expensive computationally.[1] Knowing already that the F-measure curves follow a certain shape, as we explained earlier, and in light of the fact that the analysis we perform over these curves, as will become clearer over the course of our discussion, is a preliminary step for identifying the most promising alternatives, we deemed 10 observation points to be sufficient for approximating the curves.

We compare the curves by taking the average of F-measures over the entire value range for K. The rationale for averaging is that, in lieu of knowledge about the optimal K, we would like to favor alternatives that yield the best overall accuracy across all K values. Naturally, an alternative fares better than another if it has a higher average F-measure. We compute the average F-measure for each curve by computing its *Area Under the Curve* (AUC) and normalizing the result.

In Table 3, we show for each case study the top-5 alternatives that yield the best average accuracy, i.e., alternatives with the largest (normalized) AUC. As suggested by the table, there is no alternative that is shared among all three case studies. We therefore cannot recommend a best alternative based on the information in this table alone.

To provide a general recommendation, we need to further consider the effect of individual syntactic measures, semantic

1. Such an analysis would have required us to execute clustering $770\times n/2$ times per case study, i.e., $770\times(604/2+\lceil 91/2\rceil+630/2)=510510$ times in total. Our approximate estimation of the execution time for such an experiment is more than 60 days on a conventional computer.

TABLE 3
Top-5 cluster computation alternatives.

Case	Syntactic Measure	Semantic Measure	Clustering Algorithm	Normalized AUC
Case-A	Levenstein	JCN	EM	0.485
	Levenstein	PATH	EM	0.484
	Levenstein	LIN	EM	0.482
	Levenstein	PATH	K-means	0.476
	Levenstein	RES	EM	0.476
Case-B	SoftTFIDF	NONE	EM	0.523
	SoftTFIDF	NONE	K-means	0.519
	Jaccard	NONE	EM	0.498
	Monge_Elkan	JCN	EM	0.498
	Euclidean	NONE	EM	0.493
Case-C	Levenstein	LIN	EM	0.559
	Levenstein	RES	EM	0.557
	Levenstein	PATH	EM	0.557
	Levenstein	JCN	EM	0.553
	SoftTFIDF	LIN	EM	0.547

measures and clustering algorithms on accuracy across all the alternatives. For example, a syntactic measure that is not employed in the absolute-best alternative but performs consistently well in all the alternatives where it is employed may be advantageous over one that is employed in the absolute-best alternative but also in some poor alternatives.

More precisely, we want to find similarity measures and clustering algorithms that yield good (but not necessarily the absolute-best) results, and at the same time, cause little variation. A standard way for performing such analysis is by building a regression tree [96]. A regression tree is a step-wise partitioning of a set of data points with the goal of minimizing, with respect to a certain metric, variation within partitions. Here, the data points are the 770 (cluster computation) alternatives and the metric of interest is the AUC. For each case study, the regression tree identifies at any level in the tree the most influential factor that explains the variation between the data points. In our context, there are three factors: (1) the syntactic measure, (2) the semantic measure, and (3) the clustering algorithm. Once the most influential factor is identified, the regression tree partitions the data points into two sets in a way as to minimize variations within the resulting sets.

Fig. 14 shows the regression trees for our case studies. In each node of the tree, we show the count (number of alternatives), and the mean and standard deviation for AUC. By convention, sibling nodes in the tree are ordered from left to right based on their mean values. That is, the node on a right branch has a larger mean than its sibling on the left. For every expanded (i.e., non-leaf) node, the node shows the difference between the mean values of its right and left children. For each case study, we iteratively expanded the regression tree until the difference (in means) between the right and the left nodes fell below the threshold of 0.025, which we deemed to be the minimum difference of practical relevance. Naturally, we expanded only the right branches, given the convention we noted above.

As the regression trees of Fig. 14 suggest, the most influential factor in all three case studies is the choice of clustering

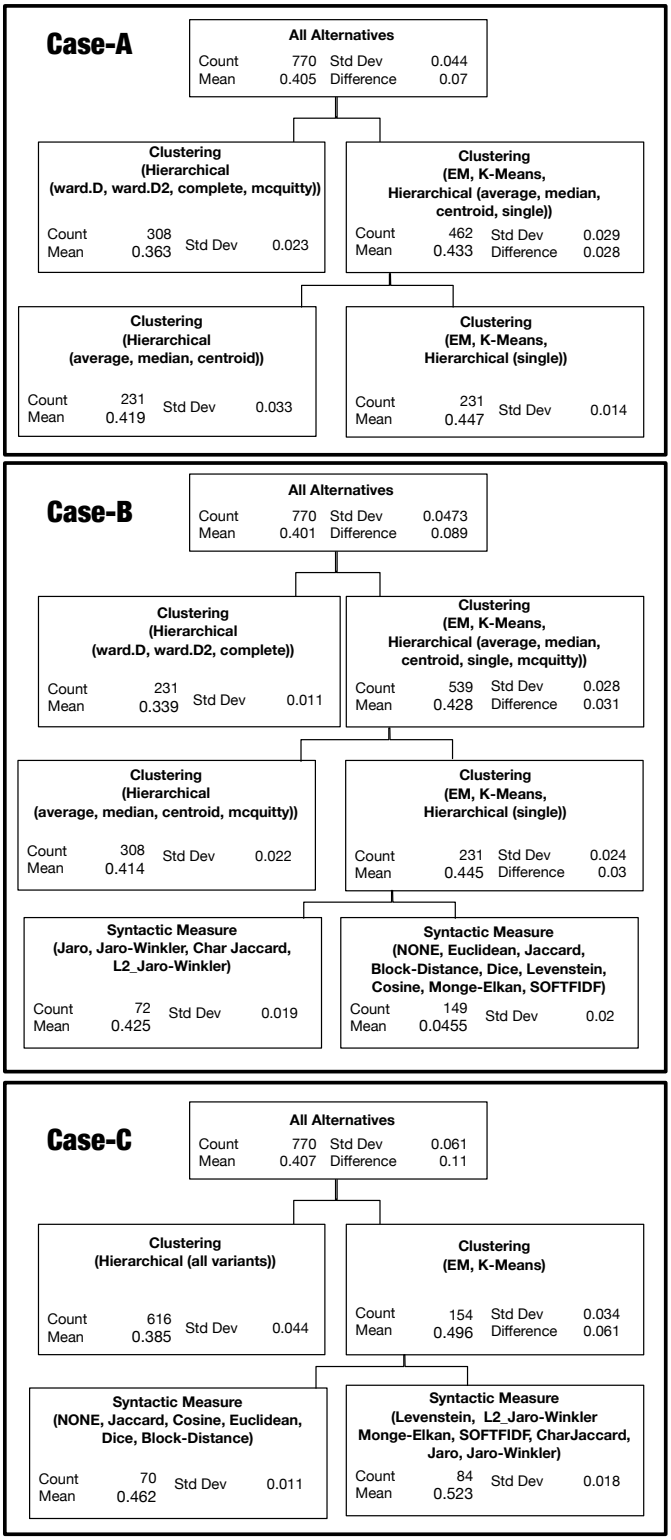


Fig. 14. Regression trees for normalized AUCs.

algorithm. Only EM and K-means perform consistently well across all our case studies, thus ruling out all variants of hierarchical clustering. For Case-A, the choice of similarity measures does not play a significant role. But, for Case-B and Case-C, the choice of syntactic measure is the second most influential factor, as shown by the regression trees of these two case studies. Taking the overlap between the best-performing

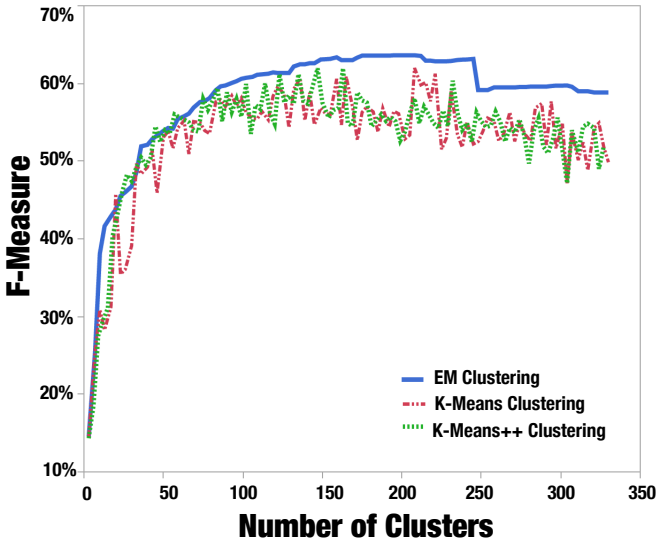


Fig. 15. Comparison between the K-means and EM clustering algorithms; both curves are for Case-C and computed using the combination of Levenstein and PATH.

syntactic measures in Case-B and Case-C, we narrow the choices of syntactic measures to Levenstein, Monge-Elkan and SoftTFIDF. The label NONE, appearing in the regression trees of Case-B and Case-C, denotes stand-alone applications of *semantic* measures (i.e., without a syntactic measure). We observe that in Case-C, NONE appears on a left branch. We thus conclude that individual semantic measures should *not* be applied on their own.

Our regression tree analysis finds both EM and K-means to be good choices for the clustering algorithm. To develop further insights into the behavior of EM and K-means, we plotted higher-resolution F-measure curves for both algorithms when applied in combination with the best similarity measures identified through our regression tree analysis. Specifically, we increased the number of points on the x-axis from 10 (used for the charts of Fig. 13) to 100.

Fig. 15 shows the higher-resolution F-measure curves for K-means and EM. The figure also shows a third curve for a variant of K-means, namely K-means++, which we will discuss momentarily. The curves in this figure differ only in the choice of the clustering algorithm. We observe from the figure that the curve for EM is relatively smooth; whereas the one for K-means has spikes. These spikes are most plausibly attributable to K-means' well-known sensitivity to the choice of initial centroids and to the number of clusters [97]. The sensitivity to the choice of initial centroids can sometimes be reduced by applying K-means++ [98], which attempts to pick the initial centroids in a way that would avoid convergence toward local optima [99].

As shown in Fig. 15, spikes are prevalent in the F-measure curve for K-means++, too. This trend of behavior was seen in both K-means and K-means++ across all our case studies and all the similarity measures considered. This observation suggests that either the initial centroids picked by K-means++ are still not good enough to steer K-means from local optima, or that the spikes are more likely to be due to sensitivity to the number of clusters. Irrespective of its root cause, the behavior seen from K-means and K-means++ is undesirable in our context, where one can never exactly pinpoint the optimal number of clusters and can only employ heuristic guidelines for coming up with a reasonable estimate (RQ3). Consequently, we rule out K-means and its variant K-means++. This narrows the choice of clustering algorithm to EM.

To summarize our discussion of RQ2, we conclude that the following similarity measures and clustering algorithm are good choices for further examination in RQ4:

> **Syntactic measures**: One of the three syntactic measures: *Levenstein*, *Monge-Elkan* or *SoftTFIDF*.
>
> **Semantic measures**: Semantic measures do not have a significant impact on clustering accuracy.
>
> **Clustering algorithm**: *EM*.

Despite semantic measures not being influential in improving clustering accuracy, we do not discourage the use of these measures. A potential reason why we did not find semantic measures useful is that semantic synonyms are not prevalent in our case studies. Semantic measures, when combined with syntactic ones, may be useful if the likelihood of (semantic) synonyms being present is high. As indicated in Section 4, our tool already supports semantic measures.

We note that statistical significance testing would offer little value in distinguishing between the choices recommended above. This is because these choices were derived from regression tree analysis. Consequently, the accuracy difference (effect size) between the different recommended choices would invariably be small, no matter whether the difference is statistically significant or not.

5.6.3 RQ3. How can one specify the number of clusters?

Theoretically, the number of clusters, K, is a value between 1 and the total number of candidate terms. However, as we argued in RQ2, considering a K value that is larger than half the number of candidate terms is unreasonable from a practical standpoint. If K is too small, accuracy will be low due to large clusters with low precision. If K is too large, accuracy will again be low but this time due to small clusters with low recall. Obviously, one does not have access to a gold standard outside an evaluation setting. Therefore, one cannot use the optimal point in F-measure curves such as those in Fig. 13 for choosing K.

We use BIC, discussed in Section 3.3, to choose a value for K. As explained in this earlier section, the intuition is that the larger the BIC, the better is the choice for the value of K. In other words, our goal should be to choose K in a way that maximizes BIC. We apply an adaptation of this general idea for choosing K, as we describe next.

For a specific cluster computation alternative, we first plot the BIC curve over the range $[1, n/2]$ of the number of clusters, where n is the number of candidate terms. To illustrate, Fig. 16 shows the BIC curves resulting from the application of EM using similarities calculated with SoftTFIDF for each case study.

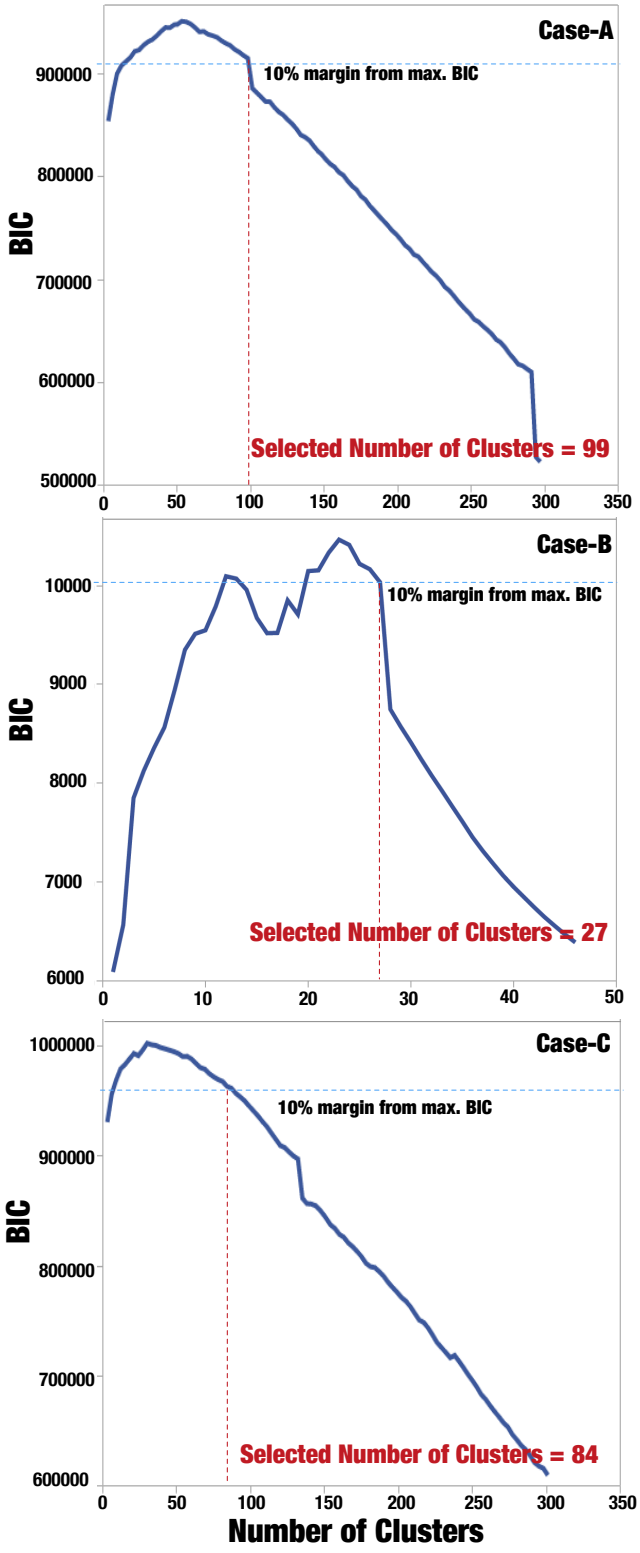


Fig. 16. Selecting the number of clusters using BIC.

For performance reasons, when the number of candidate terms (and thus the number of values in the range $[1, n/2]$) is large, we may elect not to compute the BIC at every value on the x-axis. Instead, like in RQ2, we may divide the x-axis into a certain number of points. For example, in Fig. 16, the x-axis for Case-A and Case-B is divided in increments of 1% of the range (100 points). For Case-B, the range is small (<100 points); therefore, all the values in the range are covered.

Let BIC_{max} and BIC_{min} be the maximum and minimum BIC values, respectively; and let $\mathit{margin} = (\text{BIC}_{\text{max}} - \text{BIC}_{\text{min}})/10$, i.e., 10% of the difference between BIC_{max} and BIC_{min}. Assuming that BIC_{max} occurs over a peak, we look to the right of the peak and choose the largest possible K whose BIC is larger than or equal to $\text{BIC}_{\text{max}} - \mathit{margin}$. In the (unlikely) case where the BIC curve is monotonically-increasing, i.e., there is no peak, we set K to be $n/2$. The rationale for choosing a K with a smaller BIC than the maximum is to obtain a larger number of clusters and hence a smaller number of terms within individual clusters. As long as BIC remains uncompromised, i.e., stays within a small margin from the maximum, clusters with a smaller number of terms are more desirable because they are more homogeneous and easier for analysts to inspect.

In the example curves of Fig. 16, BIC_{max} for Case-A occurs when there are 55 clusters. The BIC value nonetheless stays within the 10% margin up to 99 clusters. Based on our argument above, we select $K = 99$ for Case-A. In Case-B, BIC_{max} occurs at 23 clusters. BIC stays within the specified margin up to 27 clusters, followed by a steep decline. We therefore select $K = 27$ for Case-B. In Case-C, BIC_{max} occurs at 30 clusters but remains within the margin up to 84 clusters. Hence, in this case, we select $K = 84$.

We note that our guidelines for selecting the number of clusters are *automatable* and have been already implemented into our tool support (Section 4). Therefore, the computation of BIC and its interpretation are transparent to the end-users of our approach.

5.6.4 RQ4. Which of the alternatives identified in RQ2 are the most accurate when used with the guidelines from RQ3?

The analysis of RQ2 narrows cluster computation alternatives to one clustering algorithm, namely EM, and three syntactic measures, namely Levenstein, Monge-Elkan and SoftTFIDF. The analysis further suggests that semantic measures do not influence clustering accuracy in a major way. Nevertheless, and as discussed in RQ2, we do not rule out the usefulness of semantic measures in general. For answering RQ4, we therefore consider all the viable semantic measures from RQ2, namely LIN, PATH, RES, JCN and WUP.

Specifically, we answer RQ4 by investigating the 18 alternatives shown in Fig. 17. We leave the clustering algorithm (EM) implicit in the figure because it is the same across all these alternatives. We represent a pairwise combination of a syntactic and a semantic measure by concatenating their names separated by the "." symbol. For example, we write SoftTFIDF.PATH to refer to the combination of SoftTFIDF (syntactic) and PATH (semantic). When a syntactic measure is used on its own, we use NONE to indicate the absence of a semantic measure. For example, SoftTFIDF, when used alone, is denoted SoftTFIDF.NONE.

For each alternative, we first compute the BIC curve and apply the guidelines of RQ3 to select the number of clusters (according the BIC curve produced by that specific alternative). We then calculate the accuracy (F-measure) of the

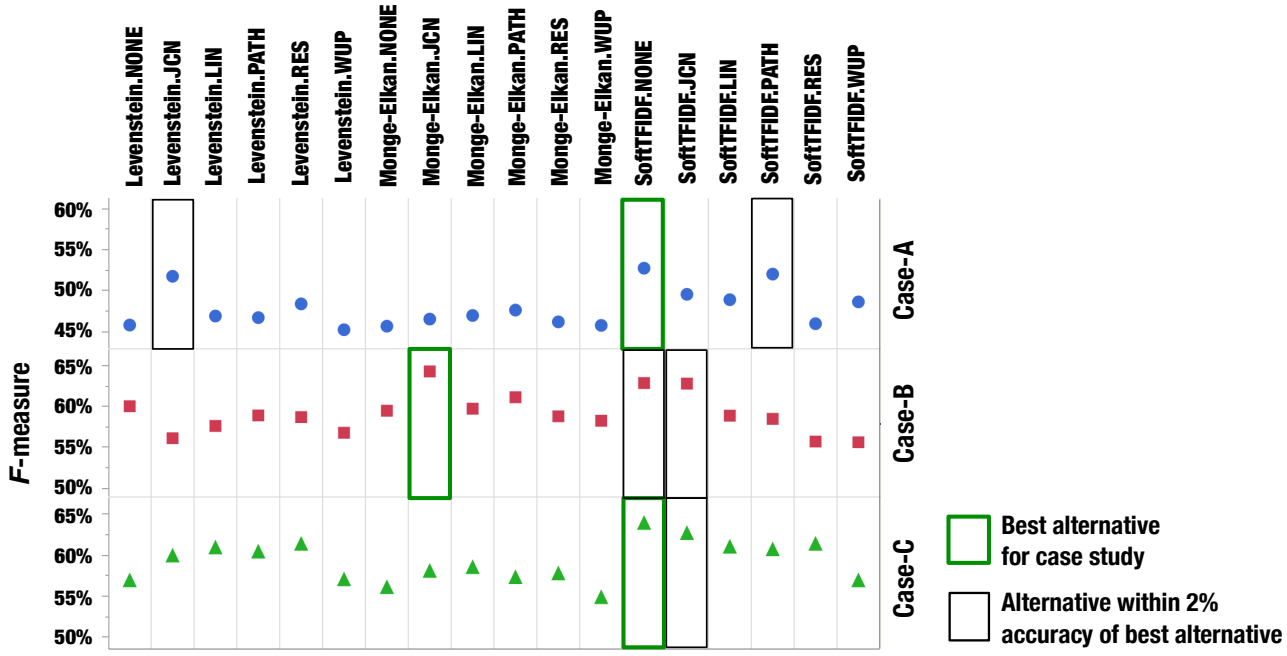


Fig. 17. Accuracy of the alternatives identified in RQ2 when the number of clusters is set using the guidelines of RQ3.

alternative at the selected number of clusters. The results are shown in Fig. 17. For each case study, we indicate both the best alternative and also any other alternative whose accuracy is within 2% of the accuracy of the best alternative.

Our results suggest that applying SoftTFIDF alone for computing similarities between terms presents the best overall option. If the analyst elects to further use a semantic measure, the best choice would be to combine SoftTFIDF with JCN: as shown by Fig. 17, this combination closely follows the stand-alone application of SoftTFIDF in terms of accuracy.

Similar to RQ2, statistical significant testing would offer little value in RQ4, since we are simply interested in picking one of the best alternatives, without taking into account whether the chosen alternative performs statistically significantly better than the other alternatives.

5.6.5 RQ5. Does our approach run in practical time?

Table 4 shows the execution times for the different steps of our approach. All execution times were measured on a laptop with a 2.3 GHz Intel CPU and 8GB of memory. The execution times we report for similarity calculation in Table 4 are based on the combined application of SoftTFIDF and JCN, i.e., the best combination from RQ4 involving a semantic measure. This provides a more realistic picture of execution times, should the application of semantic measures be warranted.

The running time of our approach is dominated by the clustering step and more specifically by the construction of the BIC curve, outlined in Section 5.6.3. For calculating a BIC curve, as was discussed earlier, we consider 1% intervals on the x-axis, instead of every possible number of clusters. The implementation we use for generating BIC curves performs, for any point on the x-axis, 10 BIC calculations corresponding to 10 different probabilistic models [31]. Each BIC value in the curve is the maximum of these 10 calculations.

We believe that the overall execution times in Table 4 are practical, noting that glossary term clustering does not need to be repeated frequently. However, handling requirements documents that are much larger that those in our case studies may require a strategy to further reduce the time spent on building BIC curves. This can be done by further limiting the number of points on the x-axis of these curves (e.g., by using larger intervals), or by considering only a subset of the probabilistic model alternatives.

5.6.6 RQ6. How effective is our clustering technique at grouping related terms?

The ideal clusters in our evaluation are overlapping. The overlaps arise because individual candidate terms may assume different roles in different requirements statements, and thus relate to potentially different terms based on each role. For example, based on the model of Fig. 9, the term "GSI monitoring information" assumes two different roles, one as a parent concept for "GSI anomaly", "GSI input parameter", and "GSI output parameter", and another as a constituent part of "Ground Station Interface". As a result, there will be two ideal clusters containing "GSI monitoring information": one ideal cluster around "GSI monitoring information" and its child concepts (terms), and another ideal cluster around the composition relationship between "Ground Station Interface" and "GSI monitoring information". By allowing overlaps in the ideal clusters, one can distinguish different roles and orient each ideal cluster around one particular role. Having the roles separated from one another is advantageous because it makes the clusters highly-focused and small.

The above said, generating clusters that are overlapping can

TABLE 4
Execution times.

Case / strategy to increment # of clusters	Phase	Execution Time
Case-A	Term Extraction	15s
380 requirements statements, 604 terms	Similarity Calculation (Syntactic + Semantic)	32s + 58s = 90s
# clusters incremented by 1% of # of terms	Clustering	20m
	Total	21m 45s
Case-B	Term Extraction	72s
110 requirements statements, 91 terms	Similarity Calculation (Syntactic + Semantic)	18s + 49s = 67s
# clusters incremented by 1 unit in each run	Clustering	15s
	Total	1m 55s
Case-C	Term Extraction	13s
138 requirements statements, 630 terms	Similarity Calculation (Syntactic + Semantic)	33s + 61s = 94s
# clusters incremented by 1% of # of terms	Clustering	23m
	Total	24m 47s

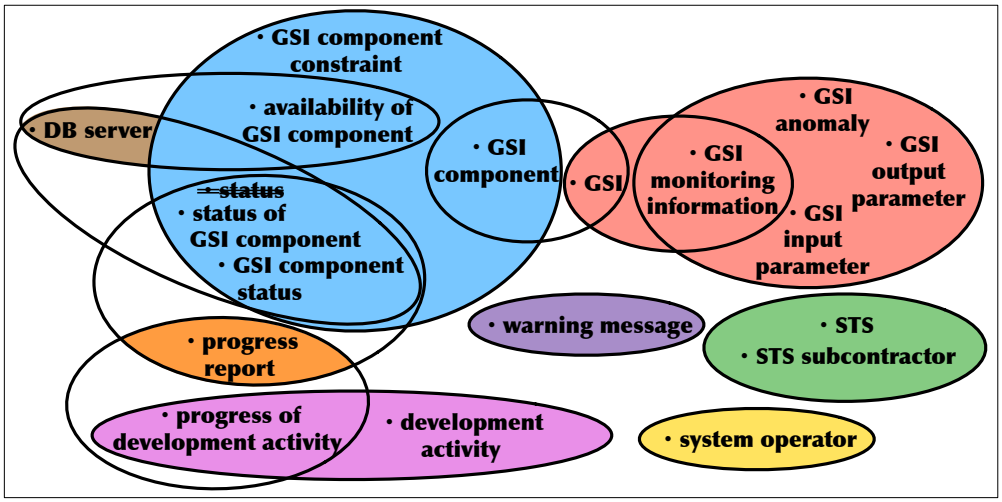


Fig. 18. Clustering example: ovals represent ideal clusters and background colors represent generated clusters.

pose an overhead for end-users, because individual candidate terms can appear multiple times in such clusters. This means that the effort associated with manually reviewing overlapping clusters will be, potentially by several folds, higher than the case where the generated clusters are non-overlapping. The clustering algorithms we considered in our work are partitioning algorithms. Nevertheless, we evaluated the results of these algorithms against overlapping ideal clusters to determine how close we can get to the (conceptually) ideal situation, without actually making the generated clusters overlapping.

To illustrate, we show in Fig. 18 both the ideal clusters and the generated ones (by EM and SoftTFIDF) for the example of Fig. 1. Here, there are 12 ideal clusters, represented as ovals, and 8 generated clusters, represented using colors. The term "status" appears in the ideal clusters but is struck out from the generated ones. This term, which is a variant of "GSI Component status" is filtered due to being a single-word common noun (see Table 1).

The clustering precision and recall for this example are 73.7% and 75%, respectively. These numbers mean that, on average, 73.7% of the terms an analyst sees in a generated cluster pertain to the specific aspect of relatedness they are investigating. Further, 75% of the ideal terms for this specific relatedness aspect have been retrieved. Despite the accuracy not being perfect, the generated clusters provide reasonable cues about related terms. The extent to which practitioners find the generated clusters useful is addressed in RQ7.

Given that achieving perfect accuracy is theoretically impossible (unless the ideal clusters have no overlaps), we need to find an upper bound on the maximum possible accuracy that one can expect from partitioning clustering in our context. This upper bound provides a reference point for evaluating the effectiveness of clustering in our approach.

To compute such an upper bound, we follow a randomized procedure. Specifically, we impose a random order on the ideal clusters and prune these clusters so that the following constraint holds for any given term t: if t appears in a cluster C at position i in the ordering, then t cannot appear in any

TABLE 5
(a) Upper bounds for the accuracy of partitioning clustering, (b) the actual accuracy of our approach.

(a) Best Possible

Case	Precision	Recall	F-Measure
Case-A	90.8%	76.2%	82.9%
Case-B	89.1%	70.7%	78.8%
Case-C	93.5%	79.1%	85.7%

(b) EM + SoftTFIDF

Case	Absolute Precision (Relative Precision)*	Absolute Recall (Relative Recall)*	Absolute F-Measure (Relative F-Measure)*
Case-A	66.8% (73.6%)	43.6% (57.2%)	52.8% (63.7%)
Case-B	79.7% (89.5%)	51.9% (73.4%)	62.9% (79.8%)
Case-C	77.4% (82.8%)	54.5% (68.9%)	64.0% (74.7%)

* *Relative = Absolute / Best Possible * 100*

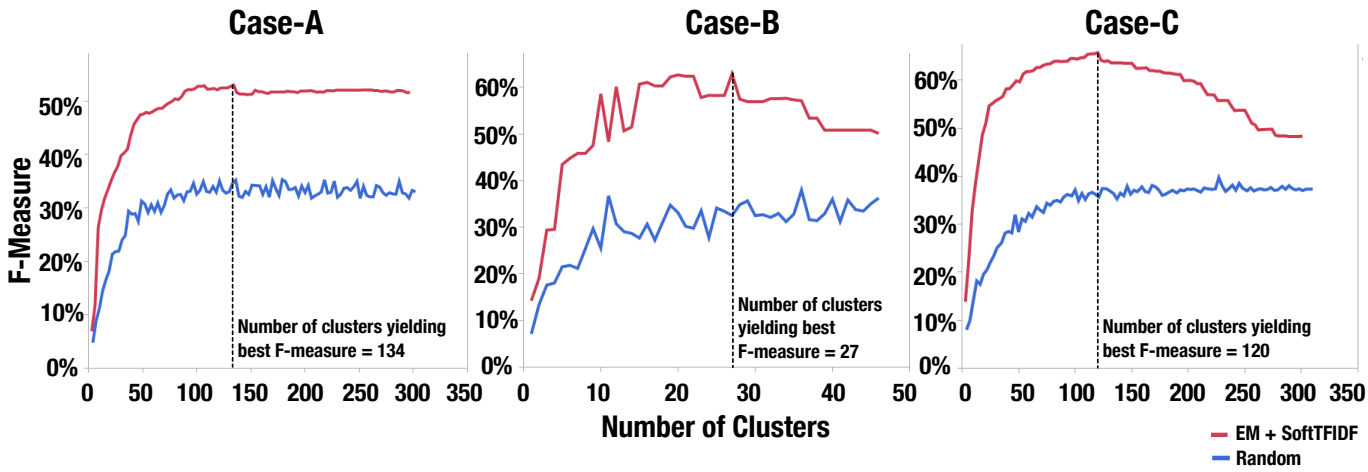


Fig. 19. Comparison with random partitioning; the optimal number of clusters for each case study is further shown.

cluster C' at a position i' such that $i' > i$. This procedure derives non-overlapping clusters from the ideal clusters. The accuracy of these non-overlapping clusters is a good indicator for what partitioning clustering can achieve in the best case. We applied the above procedure to 1000 random orders of the ideal clusters in our three case studies, and computed the average accuracy metrics.

Table 5(a) shows upper bounds for precision, recall, and F-measure when the overlaps have been removed. For comparison, we provide in Table 5(b) the accuracy of the best alternative from RQ4 (i.e., EM + SoftTFIDF), with the number of clusters chosen as discussed in RQ3. In addition, Table 5(b) shows the accuracy scores relative to the upper bounds of Table 5(a). We believe that these results, when complemented with the expert feedback discussed later in RQ7, provide positive evidence for the effectiveness of our approach.

As a sanity check, we compare our best cluster computation alternative with random partitioning. For this purpose, we use a 10-fold random partitioning of candidate terms into equal-sized clusters. In Fig. 19, we show the F-measure curve of the best alternative against that of random partitioning. As can be seen from the figure, the best alternative significantly outperforms the random baseline.

Fig. 20 shows, for each case study, the size distribution of the clusters computed using the best alternative. To facilitate comparison, we show the distributions both for the situation where we select the number of clusters based on the guidelines of RQ3, and also for the situation where the number of clusters is chosen in a way as to maximize F-measure. The number of clusters maximizing F-measure in Case-A, Case-B and Case-C are 134, 27 and 120, respectively, as marked on the charts of Fig. 19. We note that these optimal numbers of clusters are known only in an evaluation setting, i.e., when the ideal clusters are known.

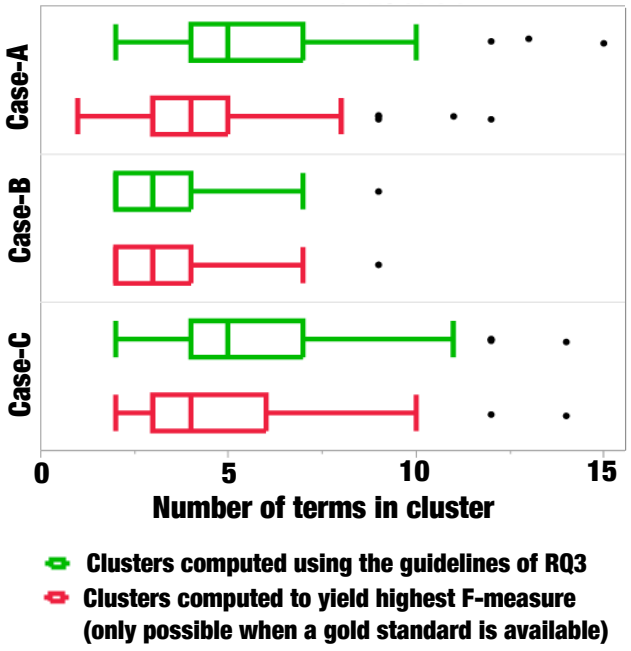


Fig. 20. Cluster size distributions.

The results in Fig. 20 provide confidence that: (1) the

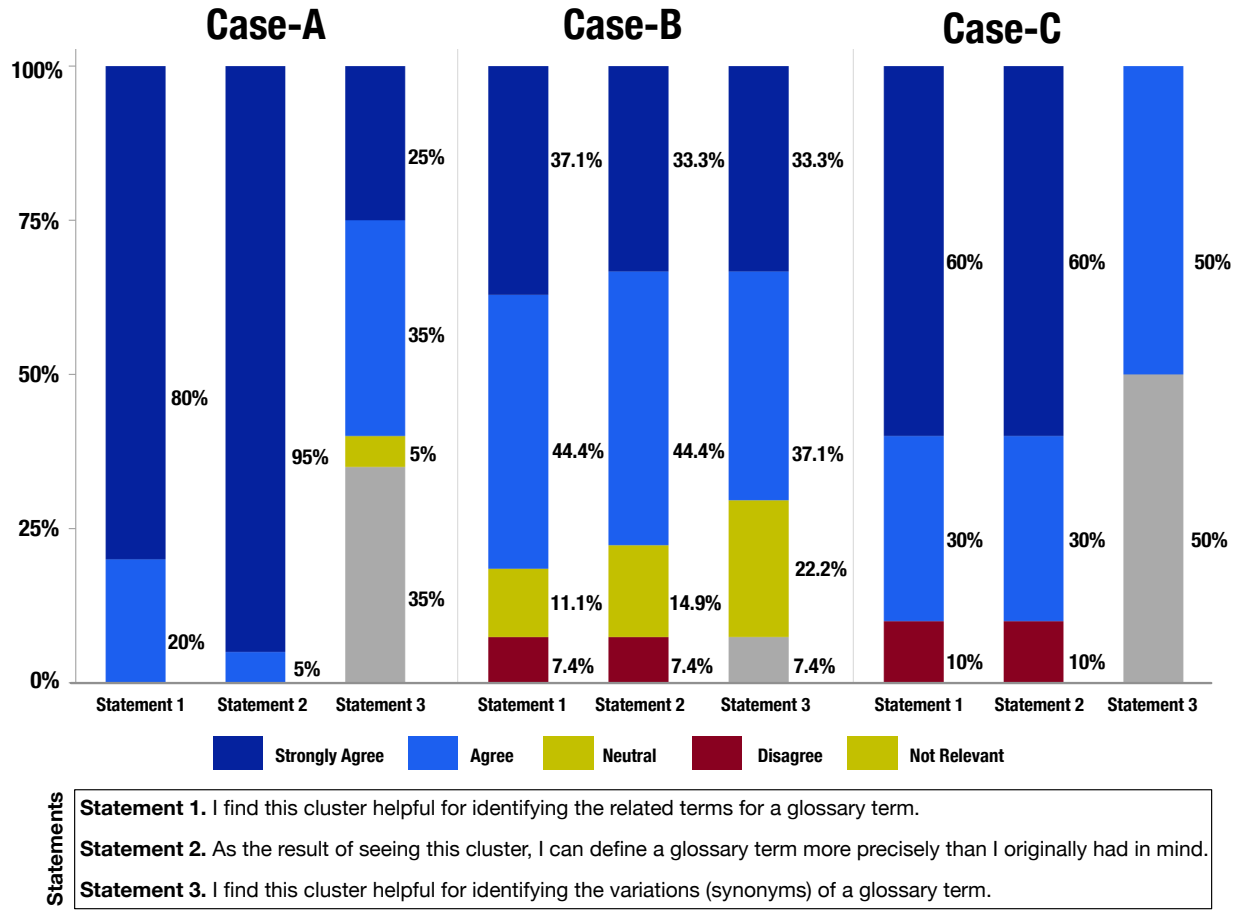


Fig. 21. Barchart representation for the expert survey interview results.

generated clusters are reasonably small and thus easy for the analysts to review; and (2) the size distributions we obtain by following the guidelines of RQ3 are not drastically different than those obtained from an optimal (but in a realistic setting, unattainable) clustering.

A final remark we need to make about the distributions of Fig. 20 is that these distributions are *not* directly comparable to those of the ideal clusters, shown earlier in Fig. 2. This is because the generated clusters cover *all* candidate terms; whereas the ideal clusters are concerned with only the actual glossary terms.

5.6.7 RQ7. Do practitioners find the clusters generated by our approach useful?

Figs. 21 and 22 summarize the results of our survey study for Case-A, Case-B and Case-C, obtained by following the procedure described in Section 5.4.3. In Fig. 21, the results are shown using barcharts for the individual case studies; whereas in Fig. 22, the results of the three case studies are combined and depicted as a heatmap [100]. Each region of this heatmap corresponds to the frequency of a certain response by the experts (Y-axis) for a given statement (X-axis). The darker a region is in the heatmap, the higher is the frequency of a certain response by the experts. We recall that the number of clusters considered in our surveys for Case-A, Case-B and Case-C are 20, 27 and 20, respectively. The heatmap is thus based on a total of 67 data points. We further note that the "Not Relevant" response does not apply to Statements 1 and 2, as marked in the heatmap.

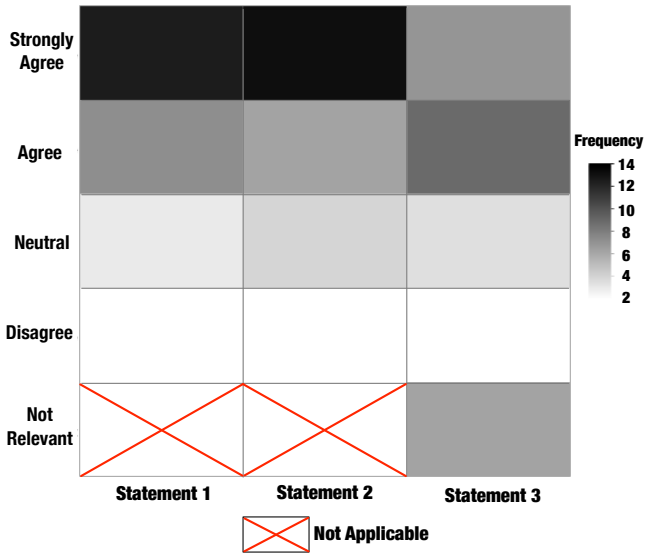


Fig. 22. Heatmap representation for the expert survey interview results.

With regard to Statement 1, the experts strongly agreed or agreed in 89.6% (60/67) of the cases that the clusters were helpful in identifying related terms. The experts were neutral in 4.4% of the cases (3 clusters, all in Case-B), and disagreed in 6% of the cases (4 clusters, two in Case-B and two in Case-C).

With regard to Statement 2, in 88% (59/67) of the cases, the experts strongly agreed or agreed that the clusters were helpful for defining the glossary terms more precisely. In 6% of the cases (4 clusters, all in Case-B), the experts were neutral; and in the remaining 6% of the cases (4 clusters, two in Case-B and two in Case-C), the experts disagreed.

Finally and with regard to Statement 3, the experts deemed 28.4% (19/67) of the clusters not relevant, meaning that the experts did not see variants in these clusters. From the remaining clusters, the experts either agreed or strongly agreed in 61.2% (41/67) of the cases that the clusters where helpful for identifying variant terms. The experts were neutral in 10.4% of the cases (one in Case-A and six in Case-B).

We compute the average of the expert responses by quantifying the agreement scale from 0 for "Strongly Disagree" to 4 for "Strongly Agree". This would give us an average of 3.40 for Statement 1, an average of 3.42 for Statement 2, and an average of 3.15 for Statement 3, noting that we exclude for Statement 3 the clusters that were deemed not relevant. The average scores for all three statements are therefore between "Agree" and "Strongly Agree".

Overall, our survey results suggest that the experts had a strong positive perception of the quality of the generated clusters.

6 Threats to Validity

In this section, we discuss threats to the validity of our empirical results and the steps we have taken to mitigate these threats.

Internal validity. The researchers were involved in the construction of the domain models from which the ideal clusters were derived. This raises the potential problem that the domain models could be built in a way that would favor our clustering results. To mitigate bias during domain model construction, we adhered to standard guidelines for domain modeling, notably by Larman [90]. Further, we subjected the domain models to a thorough review and revision process with close participation from the experts, who were familiar with domain modeling but not with the exact analytical purpose of a domain model in our evaluation.

A second potential threat to internal validity is that, as we stated in Section 5.2, in two of our case studies, Case-A and Case-B, an attempt had been made by the requirements authors to conform to a certain template. Applying a template often leads to simpler requirements sentences. This can in turn potentially reduce the error rate of NLP, thus increasing the accuracy of term extraction over template requirements when compared to non-template requirements.

We have seen no evidence of the above phenomenon happening in practice. As we discussed in Section 5.6.1, the number of false negatives caused by NLP errors is very small. Of the seven such errors across the three case studies, two occurred in Case-A, one in Case-B, and four in Case-C. When normalized by the total number of extracted terms in each case study, the NLP error rate is at 0.33% (2/604) in Case-A, 1.1% (1/91) in Case-B, and 0.63% (4/630) in Case-C. These fractions, irrespective of whether templates have or have not been used, are too small to affect accuracy in a significant way. We therefore do not anticipate the use of template requirements in our evaluation to pose a major validity threat.

Construct validity. The definition of ideal clusters is a subjective procedure. To mitigate construct validity threats, we applied an explicit and systematic process for defining the ideal clusters, building on the notion of a domain model. This limits subjectivity in defining the ideal clusters and further makes the process repeatable.

Conclusion validity. The choice of clustering accuracy metrics has an impact on the conclusions drawn based on the metrics. To minimize threats to conclusion validity, we chose the metrics in a way as to best match the overlapping nature of the ideal clusters in our problem. A complementary measure for countering conclusion validity threats is the interview survey analysis we performed in order to directly assess the usefulness of the generated clusters from a practitioner's perspective. As we explained in Section 5.4.3, we surveyed one expert per case study due to the special criteria that potential respondents had to meet, but covered multiple clusters in each survey to mitigate potential expert errors.

External validity. We applied our approach to three case studies drawn from two industry sectors. The consistency seen across the results of the case studies provides confidence about the generalizability of our results. Further case studies are nonetheless necessary for improving external validity.

7 Conclusion

We presented a tool-supported approach for extracting candidate glossary terms from natural language requirements and grouping these terms into clusters based on relatedness. One of the main advantages of our approach is that it provides guidelines on how to tune clustering for a given requirements document. This is important for a successful application of our approach in industry.

We reported on the application of our approach to three industrial case studies, in the context of which we evaluated the accuracy and usefulness of the approach. An important finding from our evaluation is that, over requirements documents, our term extraction technique is significantly more accurate than generic term extraction tools. Furthermore, the feedback we have collected from the subject matter experts in our case studies strongly suggests that our clustering technique offers practical benefits. Particularly, the experts found the resulting clusters helpful for better handling of some important tasks involved in the construction of requirements glossaries, including writing definitions for glossary terms and identifying the related terms.

In the future, we plan to incorporate a user feedback loop into our tool so that user decisions about the glossary at any

given step can be used for providing better recommendations in the subsequent steps. Another direction for future work is to conduct empirical studies aimed at determining whether our approach leads to compelling quality improvements and cost reductions in comparison with the situation where no automated assistance is used during requirements glossary construction. Finally, we would like to look into additional avenues for utilizing our approach. In particular, we intend to investigate whether our approach can be a useful decision aid for exploring the relationships between the terminologies of requirements documents that originate from different sources.

Acknowledgments. This project has received funding from Luxembourg's National Research Fund (grant agreement numbers FNR/P10/03 and FNR-6911386), and from the European Research Council under the European Union's Horizon 2020 research and innovation program (grant agreement number 694277). We are grateful to Dan Isaac, Konstantinos Liolis, Sunil Nair, and Jose Luis de la Vara for useful discussions and contributions to our case studies. We would further like to thank the anonymous reviewers of IEEE TSE for their valuable comments.

REFERENCES

[1] S. Lauesen, *Software Requirements: Styles & Techniques*, 1st ed. Addison-Wesley, 2002.

[2] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, 1st ed. Wiley, 2009.

[3] K. Schneider, *Experience and Knowledge Management in Software Engineering*. Springer, 2009, ch. Structuring Knowledge for Reuse.

[4] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*, 1st ed. Springer, 2010.

[5] R. Young, *The Requirements Engineering Handbook*, 1st ed. Artech House, 2004.

[6] K. Frantzi, S. Ananiadou, and H. Mima, "Automatic recognition of multi-word terms: the C-value/NC-value method," *International Journal on Digital Libraries*, vol. 3, no. 02, 2000.

[7] K. Heylen and D. De Hertog, "Automatic term extraction," *Handbook of Terminology*, vol. 1, no. 01, 2015.

[8] K. Barker and N. Cornacchia, "Using noun phrase heads to extract document keyphrases," in *Advances in Artificial Intelligence*, H. Hamilton, Ed. Springer, 2000, pp. 40–52.

[9] "TOPIA," last accessed: November 2015. [Online]. Available: {http://pypi.python.org/pypi/topia.termextract/}

[10] "TextRank," last accessed: November 2015. [Online]. Available: {http://github.com/ceteri/textrank/}

[11] "TermRaider," last accessed: November 2015. [Online]. Available: {http://gate.ac.uk/projects/neon/}

[12] L. Ramshaw and M. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora*, S. Armstrong, K. Church, P. Isabelle, S. Manzi, E. Tzoukermann, and D. Yarowsky, Eds. Springer, 1999, pp. 82–94.

[13] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Improving requirements glossary construction via clustering: Approach and industrial case studies," in *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'14)*, 2014, pp. 18:1–18:10.

[14] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Prentice Hall, 2009.

[15] J. Justeson and S. Katz, "Technical terminology: some linguistic properties and an algorithm for identification in text," *Natural Language Engineering*, vol. 1, no. 01, 1995.

[16] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated checking of conformance to requirements templates using natural language processing," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, 2015.

[17] W. Gomaa and A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications*, vol. 68, no. 13, 2013.

[18] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string distance metrics for name-matching tasks," in *Workshop on Information Integration on the Web (IIWeb'03)*, 2003, pp. 73–78.

[19] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge, 2008.

[20] A. Monge and C. Elkan, "Efficient domain-independent detection of approximately duplicate database records," in *2nd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'97)*, 1997, pp. 23–29.

[21] G. Hirst and D. St-Onge, "Lexical chains as representations of context for the detection and correction of malapropisms," in *WordNet: An electronic lexical database*, C. Fellbaum, Ed., 1998, pp. 305–332.

[22] J. Jiang and D. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *10th International Conference on Research in Computational Linguistics (ROCLING'97)*, 1997, pp. 19–33.

[23] C. Leacock and M. Chodorow, "Combining local context and wordnet similarity for word sense identification," in *WordNet: An electronic lexical database*, C. Fellbaum, Ed., 1998, pp. 265–283.

[24] D. Lin, "An information-theoretic definition of similarity." in *15th International Conference on Machine Learning (ICML'98)*, 1998, pp. 296–304.

[25] S. Banerjee and T. Pedersen, "Extended gloss overlaps as a measure of semantic relatedness," in *18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003, pp. 805–810.

[26] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet::similarity - measuring the relatedness of concepts," in *19th National Conference on Artificial Intelligence (AAAI'04)*, 2004, pp. 1024–1025.

[27] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995, pp. 448–453.

[28] Z. Wu and M. Palmer, "Verbs semantics and lexical selection," in *32nd annual meeting on Association for Computational Linguistics (ACL'94)*, 1994, pp. 133–138.

[29] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of variant feature specifications," *IEEE Transactions on Software Engineering*, vol. 38, no. 06, 2012.

[30] C. Aggarwal and C. Reddy, *Data clustering: algorithms and applications*, 1st ed. CRC Press, 2013.

[31] "mclust," last accessed: November 2015. [Online]. Available: {http://www.stat.washington.edu/mclust/}

[32] F. Scholz, "Maximum likelihood estimation," in *Encyclopedia of Statistical Sciences*, S. Kotz, N. Johnson, and C. Read, Eds. Wiley, 1985.

[33] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 02, 1978.

[34] A. Divakaran, *Multimedia content analysis: theory and applications*, 1st ed. Springer, 2009.

[35] M. Pazienza, M. Pennacchiotti, and F. Zanzotto, "Terminology extraction: an analysis of linguistic and statistical approaches," in *Knowledge Mining*, S. Sirmakessis, Ed. Springer, 2005, pp. 255–279.

[36] D. Bourigault, "Surface grammatical analysis for the extraction of terminological noun phrases," in *14th Conference on Computational Linguistics (COLING '92)*, 1992, pp. 977–981.

[37] S. Aubin and T. Hamon, "Improving term extraction with terminological resources," in *Advances in Natural Language Processing*, T. Salakoski, F. Ginter, S. Pyysalo, and T. Pahikkala, Eds. Springer, 2006, pp. 380–387.

[38] L. Jones, E. Gassie, and S. Radhakrishnan, "INDEX: The statistical basis for an automatic conceptual phrase-indexing system," *Journal of the American Society for Information Science and Technology*, vol. 41, no. 02, 1990.

[39] Y. Matsuo and M. Ishizuka, "Keyword extraction from a single document using word co-occurrence statistical information," *International Journal on Artificial Intelligence Tools*, vol. 13, no. 01, 2004.

[40] Z. Zhang, J. Iria, C. Brewster, and F. Ciravegna, "A comparative evaluation of term recognition algorithms," in *6th International Conference on Language Resources and Evaluation (LREC'08)*, 2008.

[41] "GATE NLP Workbench," last accessed: November 2015. [Online]. Available: {http://gate.ac.uk/}

[42] P. Drouin, "Term extraction using non-technical corpora as a point of leverage," *Terminology*, vol. 9, no. 01, 2003.

[43] C. Aguilera and D. Berry, "The use of a repeated phrase finder in requirements extraction," *Journal of Systems and Software*, vol. 13, no. 3, 1990.

[44] L. Goldin and D. Berry, "AbstFinder: a prototype natural language text abstraction finder for use in requirements elicitation," *Automated Software Engineering*, vol. 4, no. 04, 1997.

[45] D. Popescu, S. Rugaber, N. Medvidovic, and D. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," in *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, B. Paech and C. Martell, Eds. Springer, 2008, pp. 103–124.

[46] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empirical Software Engineering*, vol. 15, no. 02, 2010.

[47] L. Kof, R. Gacitua, M. Rouncefield, and P. Sawyer, "Ontology and model alignment as a means for requirements validation," in *4th IEEE International Conference on Semantic Computing (ICSC'10)*, 2010, pp. 46–51.

[48] A. Dwarakanath, R. Ramnani, and S. Sengupta, "Automatic extraction of glossary terms from natural language requirements," in *21st IEEE International Requirements Engineering Conference (RE'13)*, 2013, pp. 314–319.

[49] P. Ménard and S. Ratté, "Concept extraction from business documents for software engineering projects," *Automated Software Engineering*, 2015, to appear.

[50] G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker, "Identifying word relations in software: A comparative study of semantic similarity tools," in *16th IEEE International Conference on Program Comprehension (ICPC'08)*, 2008, pp. 123–132.

[51] S. L. Abebe and P. Tonella, "Natural language parsing of program element names for concept extraction," in *18th IEEE International Conference on Program Comprehension (ICPC'10)*, 2010, pp. 156–159.

[52] S. L. Abebe, A. Alicante, A. Corazza, and P. Tonella, "Supporting concept location through identifier parsing and ontology extraction," *Journal of Systems and Software*, vol. 86, no. 11, 2013.

[53] M. Song, I. Song, and K. Lee, "Automatic extraction for creating a lexical repository of abbreviations in the biomedical literature," in *Data Warehousing and Knowledge Discovery*, A. Tjoa and J. Trujillo, Eds. Springer, 2006, pp. 384–393.

[54] B. Daille, "Variations and application-oriented terminology engineering," *Terminology*, vol. 11, no. 01, 2005.

[55] D. Bourigault and C. Jacquemin, "Term extraction + term clustering: An integrated platform for computer-aided terminology," in *9th conference on European chapter of the Association for Computational Linguistics (EACL'99)*, 1999, pp. 15–22.

[56] A. Ferrari, S. Gnesi, and G. Tolomei, "Using clustering to improve the structure of natural language requirements documents," in *Requirements Engineering: Foundation for Software Quality*, J. Doerr and A. Opdahl, Eds. Springer, 2013, pp. 34–49.

[57] M. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *6th IEEE International Conference on Software Testing, Verification and Validation (ICST'13)*, 2013, pp. 312–321.

[58] K. Chen, W. Zhang, H. Zhao, and H. Mei, "An approach to constructing feature models based on requirements clustering," in *13th IEEE International Requirements Engineering Conference (RE'05)*, 2005, pp. 31–40.

[59] C. Duan and J. Cleland-Huang, "Clustering support for automated tracing," in *22nd IEEE/ACM international conference on Automated software engineering (ASE'07)*, 2007, pp. 244–253.

[60] A. Mahmoud, "An information theoretic approach for extracting and tracing non-functional requirements," in *23rd IEEE International Requirements Engineering Conference (RE'15)*, 2015, pp. 36–45.

[61] V. Gervasi and B. Nuseibeh, "Lightweight validation of natural language requirements," *Software: Practice and Experience*, vol. 32, no. 2, 2002.

[62] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, 2005.

[63] F. Chantree, B. Nuseibeh, A. De Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in *14th IEEE International Requirements Engineering Conference (RE'06)*, 2006, pp. 59–68.

[64] N. Kiyavitskaya, N. Zeni, L. Mich, and D. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, 2008.

[65] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering*, vol. 16, no. 3, 2011.

[66] H. Femmer, D. Méndez Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer, "Rapid requirements checks with requirements smells: Two case studies," in *1st International Workshop on Rapid Continuous Software Engineering (RCoSE'14)*, 2014, pp. 10–19.

[67] J. Misra, "Terminological inconsistency analysis of natural language requirements," *Information and Software Technology (IST)*, 2015, to appear.

[68] H. Sultanov and J. Hayes, "Application of swarm techniques to requirements engineering: Requirements tracing," in *18th IEEE International Requirements Engineering Conference (RE'10)*, 2010, pp. 211–220.

[69] S. Sundaram, J. Hayes, A. Dekhtyar, and E. Holbrook, "Assessing traceability of software engineering artifacts," *Requirements Engineering*, vol. 15, no. 3, 2010.

[70] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 3, 2012.

[71] J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions," in *Future of Software Engineering (FOSE'14)*, 2014, pp. 55–69.

[72] P. Pruski, S. Lohar, W. Goss, A. Rasin, and J. Cleland-Huang, "Tiqi: answering unstructured natural language trace queries," *Requirements Engineering*, vol. 20, no. 03, 2015.

[73] C. Arora, M. Sabetzadeh, A. Goknil, L. Briand, and F. Zimmer, "Change impact analysis for natural language requirements: An NLP approach," in *23rd IEEE International Requirements Engineering Conference (RE'15)*, 2015, pp. 6–15.

[74] B. Güldali, S. Sauer, G. Engels, H. Funke, and M. Jahnich, "Semi-automated test planning for e-ID systems by using requirements clustering," in *24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09)*, 2009, pp. 29–39.

[75] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 01, 2013.

[76] T. Yue, L. Briand, and Y. Labiche, "A systematic review of transformation approaches between user requirements and analysis models," *Requirements Engineering*, vol. 16, no. 2, 2011.

[77] V. Vidya Sagar and S. Abirami, "Conceptual modeling of natural language functional requirements," *Journal of Systems and Software*, vol. 88, no. 01, 2014.

[78] E. Holbrook, J. Hayes, and A. Dekhtyar, "Toward automating requirements satisfaction assessment," in *17th IEEE International Requirements Engineering Conference (RE'09)*, 2009, pp. 149–158.

[79] M. Adedjouma, M. Sabetzadeh, and L. Briand, "Automated detection and resolution of legal cross references: Approach and a study of Luxembourg's legislation," in *22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014, pp. 63–72.

[80] N. Zeni, N. Kiyavitskaya, L. Mich, J. Cordy, and J. Mylopoulos, "GaiusT: Supporting the extraction of rights and obligations for regulatory compliance," *Requirements Engineering*, vol. 20, no. 01, 2015.

[81] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014, pp. 153–162.

[82] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *23rd IEEE International Requirements Engineering Conference (RE'15)*, 2015, pp. 116–125.

[83] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014, pp. 183–192.

[84] "WordNet," last accessed: November 2015. [Online]. Available: {http://wordnet.princeton.edu}

[85] P. Achananuparp, X. Hu, and X. Shen, "The evaluation of sentence similarity measures," in *Data Warehousing and Knowledge Discovery*, ser. Lecture Notes in Computer Science, I.-Y. Song, J. Eder, and T. Nguyen, Eds. Springer, 2008, pp. 305–316.

[86] "Apache OpenNLP," last accessed: November 2015. [Online]. Available: http://opennlp.apache.org

[87] "SimPack: A generic Java library for similarity measures in ontologies," last accessed: November 2015. [Online]. Available: {http://www.ifi.uzh.ch/ddis/simpack.html}

[88] V. Rus, M. Lintean, R. Banjade, N. Niraula, and D. Stefanescu, "SEMILAR: The semantic similarity toolkit." in *51st Annual Meeting of the Association for Computational Linguistics (ACL'13)*, 2013, pp. 163–168.

[89] "The R project," last accessed: November 2015. [Online]. Available: {http://www.r-project.org/}

[90] C. Larman, *Applying UML and Patterns*, 3rd ed. Prentice Hall, 2004.

[91] R. Likert, "A technique for the measurement of attitudes." *Archives of psychology*, vol. 22, no. 140, 1932.

[92] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'07)*, 2007, pp. 410–420.

[93] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, "A comparison of extrinsic clustering evaluation metrics based on formal constraints," *Information Retrieval*, vol. 12, no. 4, 2009.

[94] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets," in *11th International Conference on Information and Knowledge Management (CIKM'02)*, 2002, pp. 515–524.

[95] R. C. Sprinthall, *Basic Statistical Analysis*, 9th ed. Pearson Education, 2013.

[96] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, 1st ed. Wadsworth and Brooks, 1984.

[97] B. Lantz, *Machine learning with R*, 2nd ed. Packt Publishing Ltd, 2015.

[98] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *18th annual ACM-SIAM symposium on Discrete algorithms*, 2007, pp. 1027–1035.

[99] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, 1999.

[100] G. Grinstein, M. Trutschl, and U. Cvek, "High-dimensional visualizations," in *7th Data Mining Conference (KDD'01)*, 2001, pp. 7–19.

APPENDIX A
SUPPLEMENTARY DESCRIPTIONS

Tables A.1 and A.2 respectively describe the syntactic and semantic similarity measures that we considered in our evaluation for computing similarity scores between terms. Some of the syntactic measures presented in Table A.1 are hybrid, i.e., they use a secondary measure to compute similarities between individual tokens, and then combine the resulting scores into a cumulative score. An example of a hybrid measure is SoftTFIDF, which by default uses Jaro-Winkler as a secondary measure.

The alternative criteria that we considered for computing distances between clusters in (agglomerative) hierarchical clustering are presented in Table A.3.

TABLE A.1
Description of the syntactic similarity measures considered in our empirical evaluation.

Measure	**Description**
Block Distance	Computes similarity between terms by considering them as vectors and calculating the traversal distance between the vectors in a two-dimensional plane represented by the vectors.
Cosine	Computes similarity between terms by transforming the terms into vectors and then calculating the angle between the vectors.
Dice's coefficient	Computes similarity between terms by finding the tokens that are in common and then dividing the number of common tokens by the total number of tokens in the terms.
Euclidean	Computes similarity between terms by transforming the terms into vectors and calculating the normalized difference between them.
Jaccard	Computes similarity between terms by finding the common tokens and dividing the number of common tokens by the number of tokens in the union of bags of words between the terms.
CharJaccard	A variation of Jaccard similarity that works at the level of characters rather than tokens.
Jaro	Computes similarity between terms using the number of common characters in each token of the terms.
Jaro-Winkler	An extension of Jaro similarity which combines the Jaro score with the length of the common prefix between terms.
Level-Two (L2) Jaro-Winkler	A hybrid measure that computes a normalized score for all the possible substrings (at the token level) of two terms using a secondary measure (Jaro-Winkler).
Levenstein	Computes similarity between terms based on the minimum number of character edits (insertions, deletions, and substitutions) required to transform one term into the other.
Monge-Elkan	Computes similarity between terms by matching all the individual tokens of the terms and normalizing the similarity score based on the similarity of tokens.
SoftTFIDF	Computes similarity between terms based on a secondary measure, combined with the frequency of the single-word constituents of the terms in a corpus. We use Jaro-Winkler as the secondary measure. In our context, the corpus is the set of *all* candidate terms. The intuition is that two terms are more similar if they share several single-word constituents with comparable frequencies.

TABLE A.2
Description of the semantic similarity measures considered in our empirical evaluation.

Measure	Description
HSO	Computes similarity between words by finding a short path in the is-a (vertical) and has-part (horizontal) relation chains (as specified in WordNet) that does not change direction too often. An example of an is-a relation is arm "is-a" limb. An example of a has-part relation is arm "has-part" forearm.
RES	Computes similarity between words based on the information content of the least common subsumer (LCS) of the words in an is-a hierarchy. Information content is the degree of specificity of words. For example, "car" is a more specific word than "vehicle". Therefore, "car" has a higher information content value than "vehicle". The LCS is the most specific concept that two words share as an ancestor in an is-a hierarchy. For example, the LCS of "car" and "boat" is "vehicle".
JCN	Computes similarity between words by augmenting RES (above) with the individual information content of the words.
LIN	Computes similarity between words in the same manner as JCN (above) but with a slightly modified similarity formula.
LESK	Computes similarity between words by quantifying the overlap between the different dictionary meanings of the words.
PATH	Computes similarity between words based on the shortest path between them in an is-a hierarchy.
LCH	Computes similarity between words based on the shortest path between all the meanings of the words.
WUP	Computes similarity between words based on the depth of the words and their LCS in an is-a hierarchy.

TABLE A.3
Description of the alternative criteria considered in our empirical evaluation for computing cluster distances when hierarchical clustering is applied.

Measure	Description
Single link	Computes the distance between two clusters as the least distance between any constituent terms.
Complete link	Computes the distance between two clusters as the maximum distance between any constituent terms.
Average link	Computes the distance between two clusters as the average distance between all pairs of the constituent terms of two clusters.
Centroid link	Computes the distance between two clusters as the distance between their centroids.
Median link	Computes the distance between two clusters as the distance between their medians.
Ward.D link	Computes the distance between two clusters as the sum of squared deviations from terms to centroids.
Ward.D2 link	A variant of ward.D (above).
McQuitty's link	Computes the distance between a new cluster, resulting from the merge of two existing ones, and other clusters by averaging the distances from both parts of the new cluster. The merge will take effect when the two parts as a pair have the least average distance to other clusters.