Change Impact Analysis for Natural Language Requirements: An NLP Approach

Chetan Arora, Mehrdad Sabetzadeh, Arda Goknil, Lionel C. Briand
SnT Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
{chetan.arora, mehrdad.sabetzadeh, arda.goknil, lionel.briand}@uni.lu

Frank Zimmer
SES TechCom
Betzdorf, Luxembourg
frank.zimmer@ses.com

***Abstract*—Requirements are subject to frequent changes as a way to ensure that they reflect the current best understanding of a system, and to respond to factors such as new and evolving needs. Changing one requirement in a requirements specification may warrant further changes to the specification, so that the overall correctness and consistency of the specification can be maintained. A manual analysis of how a change to one requirement impacts other requirements is time-consuming and presents a challenge for large requirements specifications.**

We propose an approach based on Natural Language Processing (NLP) for analyzing the impact of change in Natural Language (NL) requirements. Our focus on NL requirements is motivated by the prevalent use of these requirements, particularly in industry. Our approach automatically detects and takes into account the phrasal structure of requirements statements. We argue about the importance of capturing the conditions under which change should propagate to enable more accurate change impact analysis. We propose a quantitative measure for calculating how likely a requirements statement is to be impacted by a change under given conditions. We conduct an evaluation of our approach by applying it to 14 change scenarios from two industrial case studies.

***Index Terms*—Change Impact Analysis, Natural Language Requirements, Natural Language Processing (NLP).**

I. Introduction

Handling change is an essential part of Requirements Engineering (RE). In early requirements stages, the functions and characteristics of a proposed system may not be adequately known. Early requirements may thus change rapidly as knowledge about the system grows. Once the requirements mature, various other triggers for requirements change may take hold, e.g., budget and time constraints, and evolving user needs.

When a requirement undergoes some change, it is important to be able to analyze how this change impacts other requirements. We refer to this activity as *inter-requirement change impact analysis*. This type of analysis is necessary for maintaining the correctness and consistency of requirements, and is further a prerequisite for analyzing the impact of requirements changes on lower-stream artifacts that are traceable to the requirements, e.g., system design and source code [1].

This paper is concerned with inter-requirement change impact analysis for *Natural Language (NL) requirements*. NL is arguably the most common representation used for expressing software requirements [2]. A (NL) requirements specification can have hundreds and sometimes thousands of requirements statements. Manually analyzing all the statements after each change to identify the impact of the change is a laborious task, thus requiring automated assistance.

R1: *The mission operation controller shall transmit satellite status reports to the user ~~help desk~~ document repository.*

R2: *The satellite management system shall provide users with the ability to transfer maintenance and service plans to the user help desk via FTP.*

R3: *The mission operation controller shall transmit any detected anomalies to the user help desk.*

~~R4:~~ *~~The mission operation controller shall implement a configuration management database.~~*

R5: *The satellite management system shall provide a mechanism for updating user-defined parameters in the configuration database.*

R6: *The satellite management system shall authorise all updates to the telemetry configuration of a satellite before applying the changes to the satellite telemetry database.*

Fig. 1. Example requirements from a satellite control system (with changes). Added text is green and underlined. Removed text is red and struck through.

Motivating Example. We use the example of Fig. 1 to illustrate how change propagates in NL requirements. The requirements in this example have been drawn from a larger requirements specification for a satellite system, and altered to preserve confidentiality and facilitate illustration.

Suppose R1 is modified as shown, i.e., by replacing "help desk" with "document repository". The modification is merely a syntactic manifestation of the change. To properly analyze this change, one needs to consider the semantic unit(s) – primarily the phrase(s) in the statement – affected by the modification. Specifically, the change in R1 may not be meant at replacing "help desk" with "document repository", but rather to replace the noun phrase "user help desk" with "user document repository".

Another important factor to consider is that a change per se may be inadequate for determining the impact of that change. For example, the change in R1 may be explained in various ways, with each explanation leading to a different impact result. Some possible explanations are: (1) We want to globally rename "user help desk"; in this case the change in R1 affects R2 and R3. (2) We want to avoid communication between "mission operation controller" and "user help desk"; in this case, R3 is affected (the system agent being "mission operation controller"), but not R2. (3) We no longer want to "transmit satellite status reports" to "user help desk" but instead to "user document repository"; in this case, the change in R1 does not affect other

requirements. To meaningfully analyze the impact of a change, we need to be able to describe in a precise and yet practical way the conditions under which the change should propagate.

For the above change to R1 and the explanations considered, one can determine the change impact by finding, in other requirements, exact matches for the phrases involved in the change and its possible explanations. Change may further propagate through semantically-related phrases that are not exact matches or close syntactic variations. To illustrate, consider the change in R2, i.e., the addition of "via FTP". If this change is meant to indicate that all transfers to the user help desk shall be done via FTP, then the change is very likely to propagate to R1 (in its original unchanged form). Although the process verb used in R1 is "transmit" and not "transfer", the semantic relatedness between the two verbs needs to be taken into consideration for identifying the impact of change.

To increase the precision of change impact analysis, one further needs to account for the phrasal structure of requirements statements when defining relatedness. To illustrate, suppose that R4 is being removed, the explanation being that a configuration management database is unnecessary. The phrase "configuration management database" only appears in R4. However, the individual words that make up this phrase have matches in R5 and R6. In R5, "configuration" and "database" both appear in the phrase "configuration database"; and "management" appears in "satellite management system". In R6, "configuration" appears in "telemetry configuration"; "management" appears in "satellite management system"; and "database" appears in "satellite telemetry database". If one applies a mechanism based on individual words to determine how the removal of R4 impacts other requirements statements, R5 and R6 would be equally likely to be impacted. However, at the phrase level, R5 has a closer match, namely "configuration database", for the phrase of interest ("configuration management database"). To differentiate R5 and R6 in terms of the likelihood of impact, we need a relatedness measure that takes phrases into consideration.

Contributions. We propose an approach for inter-requirement change impact analysis over NL requirements. In so doing, we address several questions highlighted through the example of Fig. 1: How can we (automatically) identify the phrases in a set of requirements statements? How can we capture change at the level of phrases (and not words)? How can we express the conditions for change propagation? How can we utilize the phrasal structure of requirements to predict the impact of change? And, how can we quantify the likelihood of a requirements statement being impacted by a change?

The core enabling technology for our approach is *Natural Language Processing (NLP)*. We apply a scalable NLP technique, called text chunking [3], for extracting phrases from requirements. We use the resulting phrases as a basis for detecting change, specifying how change should propagate, and calculating likelihoods for change impact. We implement our approach in a prototype tool. We evaluate our approach using 14 change scenarios from two industrial case studies. The results suggest that our approach is accurate and practical.

While our approach can be applied wherever NL is used for expressing the requirements, the approach is strongly motivated by the situation where one has access to no reliable information other then the requirements' textual content. This situation occurs, for example, when time and cost pressures prevent the development team from building requirements models, specifying the glossary terms, or capturing the requirements dependencies in a precise manner. Our approach works directly on the text of the requirements and can thus be applied in the situation described above. The main novelty of our work is in using the phrasal structure of requirements to compensate as much as possible for the absence of models, glossaries, and dependency links.

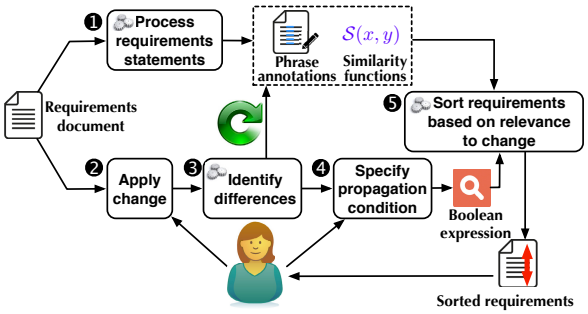

Fig. 2. Approach overview.

Structure. Section II presents an approach overview. Section III provides background information. Sections IV through VII describe the technical components of our approach. Section VIII outlines tool support. Sections IX discusses the evaluation of our approach. Section X compares the approach with related work. Section XI concludes the paper with a summary and directions for future work.

II. Overview of the Approach

The process in Fig. 2 presents an overview of our approach. The process takes as input a requirements document comprised of NL requirements statements. In Step 1, *Process requirements statements*, we apply NLP to automatically (1) identify the constituent phrases of the requirements statements and (2) compute pairwise similarity scores for all tokens (words) that appear in the identified phrases. The outputs from Step 1 are annotations delineating the phrases in the statements, and similarity functions capturing the syntactic and semantic similarities between the tokens. In Section IV, we elaborate how phrase detection and similarity calculation is performed.

In Step 2, *Apply change*, the user makes a change to the requirements document. We discuss this step in Section V. Our approach lifts proposed changes from the level of tokens to the level of phrases. This is performed in Step 3, *Identify differences*, discussed alongside Step 2 in Section V. If the change introduces phrases with words that did not exist in the requirements document before, the similarity functions produced in Step 1 will be accordingly updated.

As we argued earlier, a change by itself may not provide enough information to accurately analyze the impact of the change. In Step 4, *Specify propagation condition*, the user captures the desired condition under which the change should propagate. The propagation condition is specified in terms

of phrases, using a boolean expression. When specifying the propagation condition, the user has access to all the (automatically-detected) phrases in the original requirements document as well as any added or deleted phrases detected by Step 3. We discuss propagation conditions in Section VI. Further and as part of our tool support (Section VIII), we provide a user interface to facilitate writing these conditions.

In Step 5, *Sort requirements based on relevance to change*, the requirements statements are ordered based on the likelihood of being impacted by the change. The ordering is derived from a quantitative matching of the change propagation condition (from Step 4) against the statements. The matching is computed using the outputs of Step 1. As we elaborate in Section VII, we consider the phrasal structure of requirements statements when computing a matching.

The output from Step 5 is displayed to the user. Ideally, we would like the ordering to partition the requirements statements, with all impacted statements appearing at the beginning and all not-impacted ones at the end of the ordered set. This would help users focus on top-ranked statements with a higher likelihood of undergoing change. In Section IX, we evaluate how close our approach is to the ideal case.

We note three considerations about the process of Fig. 2: First, the process is iterative; the user can apply and analyze changes in a consecutive manner by returning to Step 2 (*Apply change*). Second, the process is interactive. This in particular requires the flexibility to make changes in real-time and obtain impact results with no, or only short, delays. Subsequently, computationally-intensive tasks, such as the calculation of similarity scores, need to be minimized over the interactive path in the process, i.e., Steps 2 through 5. The aim of Step 1 in the process is to factor out as much of these computations as possible from the interaction path. Specifically, one can run Step 1 offline and prior to change impact analysis. If a change warrants updating the outputs of Step 1, the updates are made incrementally in Step 3, as already discussed. Third, the approach allows the user to skip Steps 2–3, and start directly with articulating the change propagation condition in Step 4. This is useful when one wants to hypothesize and analyze the impact of a proposed change, say, deleting all requirements statements concerned with processing a specific object, before deciding whether or not to make the change.

III. Background

Natural Language Processing (NLP) formulates the processing rules for natural (human) language data, with the goal of getting computers to perform useful analysis over such data [3]. In this section, we introduce two specific aspects of NLP that are used in our approach: (1) detection of phrases in sentences and (2) calculation of similarity measures.

Phrase detection. We use an NLP technique known as *text chunking* for automatic detection of phrases. Text chunking decomposes a sentence into non-overlapping segments [3]. These segments can be Noun Phrases (NPs), Verb Phrases (VPs), Adjectival Phrases (ADJPs), Adverbial Phrases (ADVP), or Prepositional Phrases (PPs). Several alternative text chunking modules, called chunkers, are available within existing NLP toolkits. We previously evaluated the accuracy of some of these alternatives over requirements documents [4]. In this paper, we use the OpenNLP chunker (http://opennlp.apache.org), which, based on our evaluation, is one of the most robust alternatives. In Fig. 3, we illustrate the annotations generated by the OpenNLP chunker over an example requirements statement.

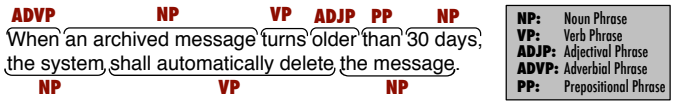

Fig. 3. Text chunking example.

For change impact analysis, we are interested in the NPs and VPs as the main meaning-bearing elements of sentences. An NP is a segment that can be the subject or object of a verb, with a possible associated adjectival modifier. A VP is a segment that contains a verb with any associated modal, auxiliary, and modifier (often an adverb). We enhance the results of text chunking with heuristics that merge adjacent NPs under certain conditions. These heuristics, developed in our earlier work [4], [5], are aimed at maintaining the semantic link between closely related NPs. For example, chunking would decompose "configuration of a satellite" into two NPs: "configuration" and "a satellite". If we treat these NPs separately, the context for "configuration" will be lost. To address this issue, we merge into a single NP any pair of adjacent NPs that match one of the following patterns: *NP of NP*, *NP's NP*, and *NP in NP*.

We note that one can use parsing, instead of chunking, to identify NPs and VPs. However, since we do not require a parse tree for our analysis, text chunking is advantageous, both in that it is computationally less expensive, and also in that it is more robust. Specifically, chunkers produce results in the large majority of cases; whereas parsers may fail to generate a parse tree, particularly when faced with unfamiliar input [3].

Similarity measures. Similarity measures for NL can be *syntactic* or *semantic*. Syntactic measures compute similarity scores based on the string content of text segments, sometimes combined with frequencies. An example syntactic similarity measure is *Levenstein similarity* [6], which computes a similarity between two strings based on the minimum number of character edits required to transform one string into the other.

Semantic measures are calculated based on correlations captured in dictionaries. An example semantic similarity measure is the *Path measure* [7], which computes a similarity score between two words based on the shortest path between them in an *is-a* hierarchy (e.g., a "car" *is-a* "vehicle" and so is a "scooter").

Most similarity measures, both syntactic and semantic, are normalized to produce a value between 0 and 1, with 0 signifying no similarity and 1 signifying a perfect match. Since there are numerous similarity measures to choose from, it is important to investigate through empirical means which measures yield the best results for a specific task.

Syntactic measures are generally best-suited for matching variations of the same word or phrase, e.g., "components of the system" and "system components", and for dealing with words

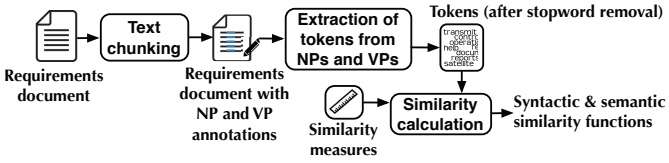

Fig. 4. Details of the requirements processing step.

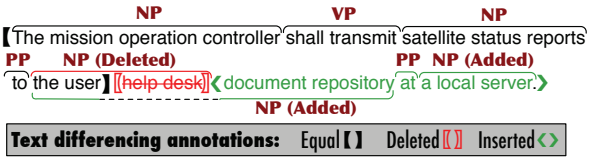

Fig. 5. Differencing example.

that are misspelled or not in dictionaries. Semantic measures are most suitable for matching words that are syntactically different but have closely-related meanings, e.g., "message" and "communication". Semantic measures further provide an accurate basis for dealing with language morphology, for instance, nominalizations, e.g., "handling" versus "handle". Due to the complementary characteristics of syntactic and semantic measures, these two classes of measures may be combined to produce higher-quality similarity scores [8].

We note that semantic measures are typically more expensive than syntactic measures to compute. Therefore, when using semantic measures, one needs to consider the level of scalability required for the task at hand. Recent advances in NLP address many of the scalability challenges associated with semantic measures. In particular, the newly-developed SEMILAR (SEMantic simILARity) toolkit [7] provides efficient implementations for a number of semantic measures.

In this paper, we experiment with several syntactic and semantic measures, and their combinations for change impact analysis. The best measures for our application context are discussed in our empirical evaluation (Section IX).

IV. Processing of Requirements

The requirements processing step, the details of which are depicted in the diagram of Fig. 4, detects the phrases in the requirements statements, extracts the tokens of these phrases, and computes similarity scores for the extracted tokens.

Requirements phrases are identified using text chunking, as we described in Section III. From the annotation produced by text chunking, we use only the noun and verb phrases (NPs and VPs). As noted earlier, these phrases are the most important to the meaning of sentences. Following text chunking, the identified NPs and VPs are broken down into their tokens to create a global, non-redundant set of tokens. We discard stopwords from this set. Stopwords are tokens that appear so frequently in the text that they lose their usefulness for text processing [9]. Stopwords include but are not limited to determiners, predeterminers, pronouns, conjunctions, and prepositions. We adapt the stopwords from the Brown corpus [10]. For example, "the user help desk" in R1 (Fig. 1) would be reduced to the following tokens: "user", "help", "desk"; "maintenance and service plans" in R2 (Fig. 1) would be reduced to: "maintenance", "service", and "plans".

Let T denote the union of all tokens extracted from the NPs and VPs of a requirements document, with the stopwords removed. We subject T to similarity calculation, as shown in Fig. 4. A (token-level) similarity function is a total function $T \times T \rightarrow [0..1]$, assigning a normalized value to every pair $(t, t') \in T \times T$ of tokens. The closer the similarity score is to 1, the more similar a pair of tokens are with respect to the similarity measure being used. We consider three alternative strategies for building a similarity function:

1) *syntactic only*, where a similarity function, $\mathcal{S}_{syn}$, is calculated using a *syntactic* measure, e.g., Levenstein similarity;
2) *semantic only*, where a similarity function, $\mathcal{S}_{sem}$, is calculated using a *semantic* measure, e.g., the path measure;
3) *combined*, where, for every pair (t, t') of tokens, we take $\max(\mathcal{S}_{syn}(t, t'), \mathcal{S}_{sem}(t, t'))$. Using max. is motivated by the complementarity of syntactic and similarity measures [8].

We zero out token similarity scores smaller than 0.3 to exclude poor token matches from further analysis. Applying this threshold is common practice when two bags of tokens are being compared based on their pairwise token similarities, e.g., see [7]. In Section VII, we use token similarities alongside phrasal information for predicting change impact. We discuss in Section IX which strategy from the three above and which similarity measures yield the most accurate results.

V. Change Application and Differencing

Applying a change (Step 2 of the process of Fig. 2) is an interactive step where the user *adds*, *deletes* or *updates* a requirements statement. To enable phrase-level analysis of changes, we cast these change operations as additions and deletions of phrases (NPs and VPs). Specifically:

- Adding (resp., deleting) a requirement amounts to adding (resp., deleting) a collection of phrases. For example, deleting R4 shown in Fig. 1 is treated as deleting the VP "shall implement" and deleting the NPs "the mission operation controller" and "a configuration management database".
- Updating a requirement amounts to a combination of phrase additions and deletions. For example, consider the requirement in Fig. 5, which is a slight extension of R1 in Fig. 1. The textual change here is deleting "help desk" and adding "document repository at a local server". This update is treated as deleting "the user help desk", and adding "the user document repository" and "a local server".

The algorithm of Fig. 6 outlines the procedure we apply for automatically detecting added and deleted phrases. The algorithm superimposes the NP and VP annotations obtained from text chunking over differencing annotations obtained from a standard text differencing tool, e.g., *diff-match-patch* [11]. Text differencing partitions a given text (in our case, a requirements statement) into regions annotated with *Equal*, *Deleted*, and *Inserted*. These annotations, illustrated in Fig. 5, denote unchanged, deleted, and inserted text segments, respectively. Given the text chunking and differencing annotations, the algorithm analyzes the overlaps between the phrases and the changed text regions, returning a set of tuples of the form (p, Op) where p is a phrase and Op is either *Added* or *Deleted*.

Input: The original requirement, R_{old}, and the changed one, R_{new};
Output: Set of phrase additions and deletions;
1: Let P_{old} be the set of phrases in R_{old}, and P_{new} that in R_{new};
2: **if** $P_{old} = \emptyset$ **then return** $\{(p, Added) \mid p \in P_{new}\}$;
3: **else if** $P_{new} = \emptyset$ **then return** $\{(p, Deleted) \mid p \in P_{old}\}$;
4: **end if**
5: Let $\ell = \emptyset$; */* Initialize the set of phrases to be returned. */*
6: **for all** $region \in$ *diff-match-patch*(R_{old}, R_{new}) **do**
7: **if** $region.type = Equal$ **then** do nothing;
8: **else if** $region.type = Deleted$ **then**
9: $\ell := \ell \cup \{(p, Deleted) \mid (p \in P_{old}) \wedge (p$ overlaps with *region*$)\}$;
10: **else if** $region.type = Inserted$ **then**
11: $\ell := \ell \cup \{(p, Added) \mid (p \in P_{new}) \wedge (p$ overlaps with *region*$)\}$;
12: **end if**
13: **end for**
14: **return** ℓ

Fig. 6. Algorithm for detecting added & deleted phrases (phrasal differencing).

For example, the output from the algorithm of Fig. 6 over the requirements change shown in Fig. 5 is the following set: {("the user help desk", *Deleted*), ("the user document repository", *Added*), ("a local server", *Added*)}. These added and deleted phrases can be used by requirements analysts for specifying the change propagation condition, discussed next.

VI. Specification of Propagation Condition

The algorithm of Fig. 6 identifies what has changed but does not explain the context and circumstances around the change. As we illustrated in Section I, an accurate analysis of the impact of a change may not be possible based solely on the change itself. To obtain meaningful results through automation, the analyst needs to make explicit any known criteria that the impacted requirements are expected to meet. Stated otherwise, for a given change, the analyst must provide a condition characterizing the requirements that the change is likely to propagate to. We refer to this condition as the *propagation condition*. Essentially, the propagation condition is the analyst's answer to the following: *What phrases do you expect to see or not to see in the requirements impacted by the change?*

We use a restricted form of boolean expressions, shown in the grammar of Fig. 7, for capturing propagation conditions. An expression is either composite or atomic (L. 1). Composite expressions are built recursively using AND and OR (L. 2). Atomic expressions can be phrases or verbatim text (L. 3). The verbatim text option is provided to support exact string search, as implemented in text editors. When phrases are indicated, a quantitative measure is applied for search to account for the syntactic and semantic variations of phrases and how the constituent words of the phrases appear in the requirements (Section VII). Phrases and verbatim text can both be negated to state they must be absent (L. 4-5). The symbols ⟨PHRASE⟩ and ⟨TEXT⟩ (L. 6, 8) are terminals. Verbatim text is enclosed in brackets (L. 8) to distinguish it from phrases (L. 6).

Table I provides examples of propagation conditions, based on the changes made to R1, R2, and R4 in the requirements of Fig. 1. In each case, we show the requirements statement number, a possible explanation for the change, and the propagation condition. In rows 1-4 and 6 of Table I, the change, as detected by the algorithm of Fig. 6, plays a role in defining the propagation condition; whereas in row 5, the newly-added

1	⟨*expression*⟩	::=	⟨*composite-expr*⟩ | ⟨*atomic-expr*⟩
2	⟨*composite-expr*⟩	::=	"(" ⟨*expression*⟩ "AND" ⟨*expression*⟩ ")" | "(" ⟨*expression*⟩ "OR" ⟨*expression*⟩ ")"
3	⟨*atomic-expr*⟩	::=	⟨*phrase*⟩ | ⟨*verbatim-text*⟩
4	⟨*phrase*⟩	::=	⟨*pos-phrase*⟩ | ⟨*neg-phrase*⟩
5	⟨*verbatim-text*⟩	::=	⟨*pos-verbatim-text*⟩ | ⟨*neg-verbatim-text*⟩
6	⟨*pos-phrase*⟩	::=	⟨PHRASE⟩
7	⟨*neg-phrase*⟩	::=	"NOT" ⟨*pos-phrase*⟩
8	⟨*pos-verbatim-text*⟩	::=	"[" ⟨TEXT⟩ "]"
9	⟨*neg-verbatim-text*⟩	::=	"NOT" ⟨*pos-verbatim-text*⟩

Fig. 7. Grammar for propagation conditions.

TABLE I
Example propagation conditions and their explanation.

	Req. (from Fig. 1)	Explanation	Propagation Condition
1	R1	Replace user help desk in subsystems X and Y only.	user help desk AND ([X] OR [Y])
2		Replace user help desk unless it is for subsystem X.	user help desk AND (NOT [X])
3		Avoid communication between mission operation controller and user help desk.	mission operation controller AND user help desk AND transmit
4		Do not transmit satellite status reports to user help desk.	satellite status report AND user help desk AND transmit
5	R2	We want to do all transfers to the user help desk via FTP.	transfer AND user help desk
6	R4	Configuration database management must be removed.	configuration database management

phrase ("FTP") does not appear in the propagation condition. This is because the propagation condition is meant to characterize what is likely to be seen in the impacted requirements, hence "FTP" not being part of the condition in row 5.

Since one cannot in general assume that analysts are comfortable with writing logical expressions, it is important to provide a more intuitive layer over logical expressions for specifying propagation conditions. To this end, we draw on findings in the Information Retrieval community, where it has been observed that people often conceptualize search queries in terms of *Conjunctive Normal Form (CNF)* expressions [12]. While our approach supports the grammar of Fig. 7 without restrictions, our tool support (Section VIII) limits propagation conditions to CNF expressions. This enables us to shield the user from logical expressions through a user interface. All the propagation conditions in Table I are in CNF.

As the examples in Table I show, propagation conditions are associated closely with *why* a certain change is being made. In our approach, we elicit propagation conditions directly from the analyst, rather than having the analyst first fully conceptualize why they made a change and then attempting to derive the propagation condition from their answer. Our decision is motivated by our experience, indicating that practitioners find it more difficult to answer why they made a change than to provide clues about how they would propagate it. This observation relates to known cognitive limitations of answering "why" questions about requirements and design rationale [13].

VII. Calculation of Impact Likelihoods

Given a propagation condition φ, we calculate for every requirements statement R a normalized matching score $\mathcal{M}(\varphi, R)$. The score is a measure of how likely R is to be impacted by the change associated with φ. The higher the score, the more likely R is to be impacted. The matching score

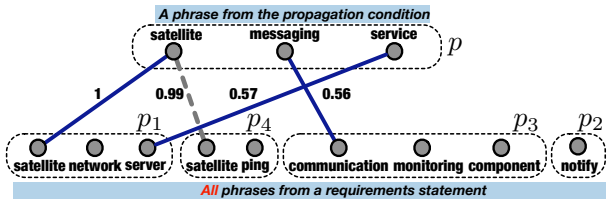

Fig. 8. Matching a propagation condition phrase against a requirement.

is computed for all requirements statements in a specification. The statements are then sorted in descending order of the score and presented to the analyst. Below, we explain how the matching scores are computed. We discuss how to use the resulting sorted list in Section IX (RQ2 and RQ3).

The matching score is computed bottom-up, as per the grammar of Fig. 7, from atomic to composite expressions. For positive verbatim text v, we evaluate $\mathcal{M}(v, R)$ to 1 if R contains v; and 0, otherwise. The core part of the matching is calculating scores for positive phrases. For positive phrase p, we calculate $\mathcal{M}(p, R)$ according to the following process:

(1) Let $[\![t_1, \cdots, t_n]\!]$ be the bag of p's tokens. Let $p_1, \cdots, p_k$ be the phrases in R; and let $[\![t'_1, \cdots, t'_m]\!]$ be the bag of the tokens of these phrases. Note that stopwords are removed from both bags. Construct a bipartite graph G, where the nodes of the first and the second parts are $[\![t_1, \cdots, t_n]\!]$ and $[\![t'_1, \cdots, t'_m]\!]$, respectively. To illustrate, suppose p is "the satellite matching service" and R is "The satellite network server shall notify the communication monitoring component after each satellite ping." The phrases in R are "The satellite network server", "shall notify", "the communication monitoring component", and "each satellite ping". In Fig. 8, we show the nodes of G. The phrases of R (and thus their constituent tokens) are ordered in a certain manner. We discuss this ordering and its use in step (2) below.

(2) We connect every pair $(t, t') \in [\![t_1, \cdots, t_n]\!] \times [\![t'_1, \cdots, t'_m]\!]$ with a weighted edge. The weight is assigned by one of the token similarity calculation strategies discussed in Section IV. Which strategy is the most accurate is discussed in Section IX (RQ1). With the edges added to G, we obtain a full weighted bipartite graph [14]. We use this graph for finding an optimal matching between the tokens of p and those of R. The optimization is cast into the assignment problem [14]. Solving the assignment problem yields a set of edges such that no two edges share the same token as an endpoint, while maximizing the sum of the weights of the selected edges. In Fig. 8, the edges of the optimal matching are shown using solid lines. The number next to each edge denotes the edge's weight. For readability reasons, we do not show G's entire edge set.

Before attempting to find an optimal match, we adjust the weights of the edges in G to nudge matching towards using the smallest possible number of phrases from R. To illustrate, consider the term "satellite" in p (Fig. 8), for which there are two exact matches in R, i.e., in phrases p_1 and p_4. Despite both p_1 and p_4 containing matches, picking "satellite" from p_1 is more sensible because at the phrase level, p_1 is a closer match to p than p_4 is. To guide matching to pick "satellite" from p_1, we rank the phrases in R according to their phrase-level similarity with p. We then levy a small penalty for picking tokens from phrases with a higher rank. More precisely, the weight of an edge between tokens t and t' is adjusted as follows: *adjusted weight* $= (1-(r-1)/100) \times$ *original weight*, where r is the rank of the phrase in which t' is located. We use a syntactic similarity measure applied to phrases to compute the ranking. Finding the best such measure for ranking is addressed in Section IX (RQ1). For instance, with Levenstein similarity applied for ranking, the phrases of R (Fig. 8) in descending order of similarity with p are p_1, p_4, p_3, and p_2. The dashed edge in Fig. 8, which is not part of the optimal matching, has an adjusted weight of $0.99 = (1-(2-1)/100)$, noting that the rank of p_4 is 2.

(3) We calculate the matching score for p as follows:

$$\mathcal{M}(p, R) = 2 \times \frac{\textit{sum of the weights of edges in optimal matching}}{N_1 + N_2 + 0.5 \times (N_3 - 1)}$$

where N_1 is the number of tokens in p, N_2 is the number of matched tokens from R, and N_3 is the number of phrases from R involved in the optimal matching. For the example of Fig. 8, the calculations are as follows: *sum of weights* $= 1.0 + 0.56 + 0.57 = 2.13$, $N_1 = 3$, $N_2 = 3$, and $N_3 = 2$ because two phrases (p_1 and p_3) contribute tokens to the optimal matching. Hence, $\mathcal{M}(p, R) = (2 \times 2.13)/(3 + 3 + 0.5(2 - 1)) = 0.66$.

The above formula is a modification of a commonly-used formula for comparing name labels [8]. In our modified formula, N_2 accounts for only the matched tokens of R rather than all its tokens. This modification prevents the matching scores from being affected by the length of requirements statements. Our modified formula further introduces an additional factor of $0.5 \times (N_3 - 1)$ in the denominator. This additional factor is motivated by the intuition that a smaller number of participating phrases from R in the optimal match makes R likely to have a stronger relationship to p. For instance, had p_1 in Fig. 8 been "satellite communication server", the optimal match would have used only p_1 from R. This would have been rewarded by increasing the matching score from 0.66 to $(2 \times 2.13)/(3 + 3 + 0.5(1 - 1)) = 0.71$.

For negated atomic and composite expressions in the grammar of Fig. 7, we calculate the matching scores as follows:

- $\mathcal{M}(\text{NOT } z, R) = 1 - \mathcal{M}(z, R)$; z is a phrase or verbatim text
- $\mathcal{M}(\varphi_1 \text{ AND } \varphi_2, R) = \mathcal{M}(\varphi_1, R) \times \mathcal{M}(\varphi_2, R)$
- $\mathcal{M}(\varphi_1 \text{ OR } \varphi_2, R) = \max(\mathcal{M}(\varphi_1, R), \mathcal{M}(\varphi_2, R))$

VIII. Tool Support

We have implemented our approach in a prototype tool, NARCIA (NAtural language Requirements Change Impact Analyzer). In Fig. 9, we present two screenshots of the tool. Fig. 9(a) shows the interface for specifying propagation conditions. The tool restricts these conditions to CNFs, as discussed in Section VI. Each box marked with a $\star$ on the screenshot of Fig. 9(a) is a CNF clause, i.e., a disjunction of phrases and possibly verbatim text items. The user can define as many clauses as necessary using the "Add" button.

To assist users in writing propagation conditions, the tool presents the original and changed requirements statements alongside the added and deleted phrases detected via phrasal

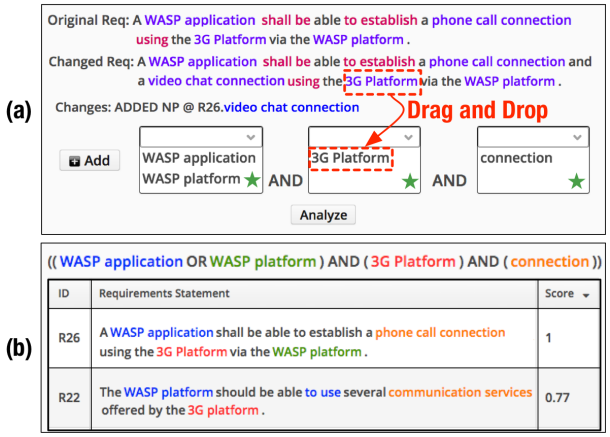

Fig. 9. NARCIA's user interface for (a) specifying propagation conditions, and (b) reviewing change impact analysis results (tool's output has been truncated).

differencing. The user can drag and drop any phrase from this information into the clause boxes, as illustrated in Fig. 9(a). For convenience, the tool automatically removes determiners and predeterminers from phrases. The drop-down lists in the clause boxes are populated with the phrases of the underlying requirements document to allow easier access to these phrases. In addition, the user has the option of directly typing in phrases or verbatim text into the clause boxes. Once the propagation condition has been provided and the "Analyze" button pressed, the tool produces a sorted list of requirements statements based on how well the statements match the provided condition. Fig. 9(b) shows a (truncated) example of the tool's output. The propagation condition is shown on the top of the result page as a logical expression.

NARCIA is implemented in Java and is approximately 8K lines of code, excluding comments and third-party libraries. For more details, see: https://sites.google.com/site/svvnarcia/.

IX. Evaluation

In this section, we investigate, through two industrial case studies, the following Research Questions (RQs):

RQ1. Which similarity measures are best suited to our approach? The choice of similarity measures used for calculating impact likelihoods has a direct bearing on the quality of the results. RQ1 aims to identify the syntactic and semantic similarity measures that lead to the most accurate results.

RQ2. How should analysts use the sorted requirements list produced by our approach? For our approach to be useful, analysts need to determine how much of the sorted list is worthwhile inspecting. In other words, they need to determine a point in the list beyond which the remainder of the list is unlikely to contain impacted requirements. The aim of RQ2 is to develop systematic guidelines on how to choose this point.

RQ3. How effective is our approach? Assuming that the guidelines resulting from RQ2 are followed, RQ3 aims to determine if our approach can reliably identify the impact set and at the same time save substantial inspection effort.

RQ4. How scalable is our approach? RQ4 aims to establish if our approach has a reasonable execution time.

TABLE II
Case Studies Used in the Evaluation.

Case	Description	# of requirements	# of phrases	# of distinct tokens	# of change scenarios
Case-A	Simulator module for a satellite ground station	160	673	648	9
Case-B	3G mobile service platform	72	267	263	5

Table II outlines key information about our case studies. The first case, hereafter called Case-A, concerns a proprietary requirements document for a satellite software component that is under development by our industry partner, SES TechCom. The second case, hereafter called Case-B, is based on a public requirements document for a context-aware mobile service platform [15]. Both documents represent real systems and were written by practitioners. Data collection for Case-A was performed as part of our current work; whereas, Case-B is drawn from the evaluation material of a previous, and different, impact analysis technique [16], [17]. For each case, we provide in Table II a brief description, the number of requirements statements, the number of phrases and distinct tokens, and the number of change scenarios considered. The source material for Case-B is available on our tool's website at http://sites.google.com/site/svvnarcia/.

The change scenarios in Case-A are *real* and based on the change history of the underlying document. We identified 14 scenarios, of which we use only 9 in our evaluation due to reasons described in the next paragraph. In Case-B, there are 5 scenarios, which are *hypothetical* but validated with the experts in Case-B in terms of being meaningful [17]. In both cases, the impact sets for the changes were provided by the experts involved in writing the requirements. Table III shows the size of the impact set for each change scenario along with the shape of the propagation condition.

TABLE III
Shapes of Propagation Conditions and Sizes of Impact Sets.

	Scenario	Propagation Condition Pattern	Size of Impact Set
Case-A	A.1	⟨NP⟩ AND ⟨NP⟩	4
	A.2	⟨NP⟩ OR ⟨NP⟩	8
	A.3	⟨NP⟩	39
	A.4	(⟨NP⟩ OR ⟨NP⟩) AND ⟨NP⟩	5
	A.5	⟨NP⟩ OR ⟨NP⟩	10
	A.6	⟨NP⟩ AND ⟨NP⟩	3
	A.7	⟨NP⟩ AND ⟨NP⟩	7
	A.8	⟨NP⟩ OR ⟨NP⟩	5
	A.9	⟨verbatim-text⟩ AND ⟨NP⟩	3
Case-B	B.1	⟨NP⟩ AND ⟨NP⟩	2
	B.2	⟨NP⟩	9
	B.3	⟨NP⟩ AND ⟨NP⟩AND ⟨NP⟩	1
	B.4	⟨NP⟩ AND ⟨NP⟩	1
	B.5	(⟨NP⟩ OR ⟨NP⟩) AND (⟨NP⟩ OR ⟨NP⟩)	9

In Case-A, the propagation conditions were specified directly by the lead engineer (fifth author). Phrasal search was used in only 9 (out of the 14) scenarios. The other 5 scenarios involved only verbatim text search and were consequently excluded due to their limited usefulness in our evaluation. In Case-B, we did not have access to the experts for specifying the propagation conditions. The first three authors independently wrote the conditions and then reached consensus on them. To avoid validity threats, the phrases in the propagation conditions of Case-B were limited to the phrases that appeared in the changed requirements statements (pre- and post-change) as well as the brief description of change rationale available in the existing documentation [17].

Next, we discuss our RQs based on Case-A and Case-B:

RQ1. To answer this RQ, we define a notion of accuracy for the sorted lists produced as described in Section VII. We do so using charts that show the percentage of impacted requirements identified (Y-axis) against the percentage of requirements traversed in the list (X-axis). In Fig. 10, we provide charts for two sorted lists (of the same requirements set), computed by two different combinations of similarity measures.

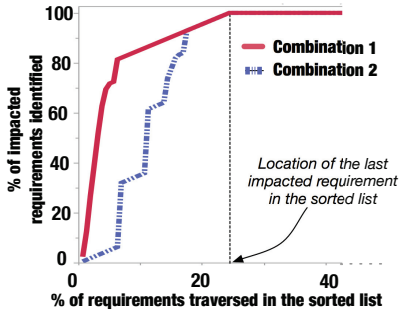

Fig. 10. Accuracy of lists produced by two combinations of similarity measures.

A simple accuracy metric would be the percentage of requirements traversed in a sorted list to identify *all* impacted requirements. While intuitive, this metric cannot distinguish the combinations in Fig 10. In both combinations, all the impacted requirements are identified after traversing 24% of the lists. Nevertheless, Combination 1 is a better alternative as it produces better results *earlier*. To reward earlier detection of impacted requirements, we use the *Area Under the Curve (AUC)* for evaluating accuracy. AUC can tell apart the combinations in Fig. 10, as the metric is larger for Combination 1.

We instantiated our approach using pairwise combinations of 10 syntactic measures from SimPack (http://www.ifi.uzh.ch/ddis/simpack.html) and 9 semantic measures from SEMILAR, including alternatives where only a syntactic or only a semantic measure is applied (i.e., where a constant zero function is used for either semantic or syntactic similarity). This yields $(10+1)\times(9+1)-1=109$ combinations. For the phrase ranking stage in the process described in Section VII, we used the syntactic similarity measure in a given combination, or Levenstein similarity when only a semantic measure was being applied for tokens. We ran these combinations on all our change scenarios, calculating the AUCs. Let S be the sum of the AUCs for the change scenarios in either Case-A or Case-B, resulting from a particular combination. The combination that maximizes S is deemed the best for the respective case study.

We obtain the best result for Case-A when *Levenstein similarity* –a syntactic measure– is applied alone, and for Case-B when *Path* –a semantic measure– is applied alone. This discrepancy is explained as follows: In Case-A, the requirements were written by a small group of engineers, and the propagation conditions were specified directly by one of these engineers. In contrast, the requirements in Case-B were written by a consortium of companies and the propagation conditions were written by people other than those involved in writing the requirements. Our analysis indicates that using semantic similarities is important for handling the heterogeneity seen in Case-B. The absolute bests in Case-A and Case-B are followed closely by the combination of Levenstein and Path. In Case-A, this combination is distinguished by only 1% from when

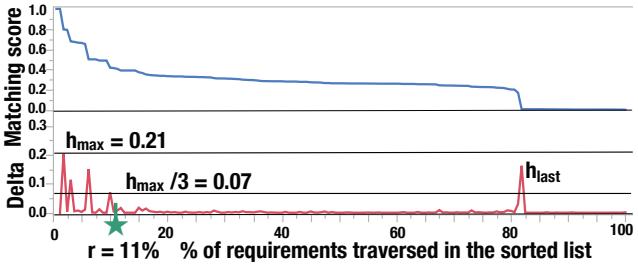

Fig. 11. Delta chart for identifying the cutoff (r).

Levenstein is applied alone (in terms of the sum of the AUCs); and, in Case-B, the difference between the combination of Levenstein and Path versus Path alone is only 2%. Given these negligible differences and the complimentary nature of syntactic and semantic measures, *we recommend the combined use of Levenstein similarity and Path*. These measures were briefly introduced in Section III.

Overall, our recommended combination yields results that, when compared to results from other combinations, are better (in terms of AUC) by an average of 11% in Case-A and 4% in Case-B. The largest accuracy difference between our recommended combination and other combinations was over a change scenario (A.9) in Case-A, where our recommended combination outperformed another by 33%. The remaining RQs are answered based on our recommended combination.

RQ2. Outside an evaluation setting, one cannot know a priori how much of a sorted list needs to be inspected before all the impacted requirements statements have been seen. To decide how much of a sorted list is worthwhile inspecting, we define a notion of *cutoff*. This notion is defined based on a delta chart, an example of which is shown in Fig. 11 for one of the change scenarios of Case-A. In the chart, at any position i on the X-axis, the Y-axis is the difference between the matching scores (impact likelihoods) at positions i and $i-1$. For easier understanding, we further show, on the top of the figure, the matching scores. We set the cutoff to be the point on the X-axis after which there are no significant peaks in the delta chart. Intuitively, the cutoff is the point beyond which the matching scores no longer adequately distinguish the requirements in terms of being impacted. What constitutes a significant peak is relative. Based on our experimental experience, a peak is significant if it is larger than one-third of the highest peak in the delta chart, denoted $\mathbf{h}_{\text{max}}$ in Fig. 11. The only exception is the peak caused by zeroing out token similarities smaller than 0.3 (see Section IV). This peak, if it exists, is always the last one and hence denoted $\mathbf{h}_{\text{last}}$. Since $\mathbf{h}_{\text{last}}$ is a mathematical artifact, it is discarded when the cutoff is being determined.

More precisely, we define the cutoff r to be at the end of the right slope of the last significant peak (excluding $\mathbf{h}_{\text{last}}$). In the example of Fig. 11, $\mathbf{h}_{\text{max}}=0.21$. Hence, r is at the end of the last peak with a height $>\mathbf{h}_{\text{max}}/3=0.07$. *We recommend that analysts should inspect the requirements statements up to the cutoff and no further.* In the example of Fig. 11, the cutoff is at 11% of the sorted list. It is important to note that the cutoff is *automatically computable*. Hence, the delta chart and its interpretation are transparent to the users.

RQ3. We answer this RQ based on the cutoff from RQ2. To be effective, our approach must produce a small number of *False Positives (FP)* and *False Negatives (FN)*. An FP is a non-impacted requirements statement that appears before the cutoff and is thus subject to manual inspection. Formally, the set of all FPs is given by $\mathcal{R}_{\text{inspected}} \setminus \mathcal{R}_{\text{impacted}}$, where $\mathcal{R}_{\text{inspected}}$ is the set of inspected requirements (for a specific change) and $\mathcal{R}_{\text{impacted}}$ is the set of impacted requirements (by that change). An FN is an impacted requirements statement that appears after the cutoff and is thus missed. The set of all FNs for a given change is $\mathcal{R}_{\text{impacted}} \setminus \mathcal{R}_{\text{inspected}}$.

Fig. 12 shows, for each change scenario in Case-A and Case-B, the percentage of FPs in terms of the total number of requirements in the respective case. In Case-A, the FP rate is between 1% and 7%, and in Case-B – between 6% and 8%, except for an outlier (scenario B.2) with an FP rate of ≈45%. Upon further investigation, we concluded that the propagation condition associated with the outlier was too unspecific, resulting in the condition to match a large number of (irrelevant) requirements. As stated earlier, due to lack of access to the experts in Case-B, we followed a conservative approach when writing the propagation conditions, using phrases only from the changed requirement and the documented change rationale. In the case of the outlier, the rationale that was available to us was not precise enough, limiting us to use a single phrase as the propagation condition. An expert would most likely have refined the propagation condition upon seeing the sorted list; however, we did not consider such a feedback loop.

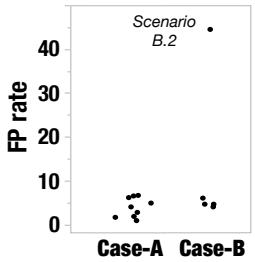

Fig. 12. FP rates.

With regards to FNs, we detected *all* the impacted requirements in Case-A, and thus an FN rate of 0 for the change scenarios in this case. The domain expert involved in the case found the impact analysis results to be very useful, considering the low FP rates (Fig. 12). For Case-B, the FN rate is 0 for all scenarios, but one (scenario B.5), where we miss 1 out of the 9 impacted requirements. The missed requirement was due to a relationship that our approach cannot identify. Specifically, the relationship is between the phrase "touristic attraction" in the missed requirement and the phrase "point of interest" in the propagation condition. The syntactic and semantic similarity measures that our approach is built upon cannot detect the tacit *is-a* relationship between the former and latter phrase. This is a limitation in our approach and needs to be addressed through further user input, e.g., a domain model.

RQ4. Table IV shows the execution times for the main computational tasks in our approach. All tasks except sorted list generation are one-offs and performed in Step 1 of the process of Fig. 2. Given the small execution times, particularly for sorted list generation, we expect our approach to scale to larger requirements documents. Execution times were measured on a laptop with a 2.3 GHz CPU and 8GB of memory.

TABLE IV
EXECUTION TIMES.

Case	Task	Execution Time
Case-A 160 requirements 673 distinct phrases 648 distinct tokens	Phrase detection (full document)	15s
	Syntactic similarity calculation	22s
	Semantic similarity calculation	208s
	Sorted list generation (avgerage)	13s
	Sorted list generation (worst case)	16s
Case-B 72 requirements 267 distinct phrases 263 distinct tokens	Phrase detection (full document)	12s
	Syntactic similarity calculation	8s
	Semantic similarity calculation	74s
	Sorted list generation (avgerage)	9s
	Sorted list generation (worst case)	11s

Threats to validity. The lack of access to experts for specifying the change propagation conditions in Case-B poses a threat to internal validity. As discussed earlier, we mitigated this threat by having three researchers independently write the propagation conditions and limiting the choice of phrases that could be used in the conditions. Another threat to internal validity is that the expert who provided the propagation conditions in Case-A is one of the authors. To alleviate bias, the expert had no exposure to the developed tool until after completing the specification of the propagation conditions.

We have evaluated our approach using two case studies in different domains. The consistency of the results across these two case studies makes us optimistic about the generalizability of our approach. Nonetheless, further case studies remain essential for minimizing threats to external validity.

X. RELATED WORK

Below, we compare our approach with related work on change impact analysis and on the application of NLP in RE.

Change Impact Analysis (CIA). Our work focuses on inter-requirement CIA, i.e., how requirements changes affect *other requirements*. We thus narrow our discussion to work strands that address this specific facet of CIA. Inter-requirement CIA requires some notion of dependency between requirements for change propagation. Zhang et al. [18] study two requirements dependency models, the *D*-model [19] and the *P*-model [20], in terms of suitability for inter-requirement CIA. They propose an enhanced model with generic dependency types such as "refines", "conflicts", and "constrains". Goknil et al. [16] propose a similar dependency model, augmenting it with formal semantics and using it for impact analysis over NL requirements. When the requirements are expressed as models, more specialized dependency types may be used. Amyot [21] uses operationalization dependencies between use cases and goals to propagate change between intentional and behavioral requirements; and Cleland-Huang et al. [22] use soft goal dependencies to analyze how changes in functional requirements impact non-functional requirements.

Our work differs from the above in that it does not need the requirements dependencies to be specified ahead of time. In our context, whether there is a dependency between a changed requirement and another requirement is determined by the propagation condition. Since one cannot enumerate all possible

conditions, building an explicit dependency graph is infeasible. Another difference in our approach is that it does not attempt to characterize the dependencies using a dependency model. The high expressiveness and implicit semantics of NL often make it difficult to classify dependencies using predefined types. Instead of using typed dependencies, we use similarity scores to assess the impact of changes.

A large body of work exists on automated retrieval of requirements trace links [23], [24], [25]. Our work takes inspiration from and follows a similar process to Just-In-Time (JIT) techniques for trace retrieval [26]. The main contribution of our work over existing JIT trace retrieval techniques is that we take the phrasal structure of requirements into consideration.

NLP in RE. NLP techniques such as tokenization, part-of-speech tagging, and similarity measurement that are used in our approach are also commonly used in, among other tasks, inconsistency and ambiguity handling, e.g., [27], [28], requirements tracing, e.g., [24], [25], and requirements overlap detection, e.g., [29]. Text chunking too has been applied in RE before, albeit to a limited extent, e.g., for matching requirements to web-service descriptions [30], assessing requirements satisfaction [31], ambiguity resolution [28], checking conformance to templates [4], and extracting requirements glossary terms [5]. We use NLP to address a different problem than those addressed by the above work.

XI. Conclusion

We presented an approach based on Natural Language Processing for analyzing the impact of changes in natural language requirements specifications. The key characteristic of our approach is that it exploits the phrasal structure of requirements sentences for analysis. Our evaluation suggests that our approach is accurate and practical in industrial settings.

An important hypothesis in our approach is that the large majority of requirements dependencies manifest themselves within the constituent phrases of requirements sentences. Across the change scenarios in our case studies, we could detect 99% (105 / 106) of the impacted requirements through phrasal analysis. Nevertheless, certain dependencies, an example of which was reported in our evaluation, are inherently tacit and detectable only through explicit guidance. In the future, we plan to extend our approach with a cost-effective mechanism for explicating such dependencies through a domain model. We further plan to enhance our approach with means for handling simultaneous changes. Finally, we plan to conduct usability studies to better validate our approach. The primary focus of these future studies will be on determining whether the change propagation conditions in our approach can be elicited effectively from practitioners.

Acknowledgment. We gratefully acknowledge funding from Luxembourg's National Research Fund (FNR/P10/03 - Verification and Validation Laboratory, and FNR-6911386).

References

[1] P. Jönsson and M. Lindvall, "Impact analysis," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer, 2005.

[2] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, 2010.

[3] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing*. Prentice Hall, 2000.

[4] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, and R. Gnaga, "Automatic checking of conformance to requirement boilerplates via text chunking: An industrial case study," in *ESEM'13*, 2013.

[5] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Improving requirements glossary construction via clustering: Approach and industrial case studies," in *ESEM'14*, 2014.

[6] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, 2008.

[7] V. Rus, M. Lintean, R. Banjade, N. Niraula, and D. Stefanescu, "SEMILAR: The semantic similarity toolkit." in *ACL*, 2013.

[8] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and merging of variant feature specifications," *IEEE TSE*, vol. 38, no. 6, 2012.

[9] C. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[10] W. Francis and H. Kucera, *Frequency analysis of English usage*. Houghton Mifflin, 1982.

[11] "Diff, match and patch libraries for plain text," http://code.google.com/p/google-diff-match-patch/.

[12] A. Pirkola, H. Keskustalo, and K. Järvelin, "The effects of conjunction, facet structure, and dictionary combinations in concept-based cross-language retrieval," *IR*, vol. 1, no. 3, 1999.

[13] A. Dutoit, R. McCall, I. Mistrik, and B. Paech, *Rationale Management in Software Engineering*. Springer, 2006.

[14] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[15] P. Ebben, "Requirements for the WASP application platform," Telematica Instituut, Tech. Rep. WASP/D2.1, 2002.

[16] A. Goknil, I. Kurtev, K. van den Berg, and W. Spijkerman, "Change impact analysis for requirements: A metamodeling approach," *IST*, vol. 56, no. 8, 2014.

[17] A. Goknil, R. van Domburg, I. Kurtev, K. van den Berg, and F. Wijnhoven, "Experimental evaluation of a tool for change impact prediction in requirements models," in *MoDRE'14*, 2014.

[18] H. Zhang, J. Li, L. Zhu, D. Jeffery, Y. Liu, Q. Wang, and M. Li, "Investigating dependencies in software requirements for change propagation analysis," *IST*, vol. 56, no. 1, 2014.

[19] Å. Dahlstedt and A. Persson, "Requirements interdependencies: State of the art and future challenges," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds. Springer, 2005.

[20] K. Pohl, *Process-Centered Requirements Engineering*. Wiley, 1996.

[21] D. Amyot, "Introduction to the user requirements notation: learning by example," *Computer Networks*, vol. 42, no. 3, 2003.

[22] J. Cleland-Huang, R. Settimi, O. B. Khadra, E. Berezhanskaya, and S. Christina, "Goal-centric traceability for managing non-functional requirements," in *ICSE*, 2005.

[23] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Computer*, vol. 40, no. 6, 2007.

[24] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case study," *IJSEKE*, vol. 22, no. 3, 2012.

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

[26] J. Cleland-Huang, "Traceability in agile projects," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012.

[27] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *TOSEM*, vol. 14, no. 3, 2005.

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

[29] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE TSE*, vol. 39, no. 1, 2013.

[30] K. Zachos and N. Maiden, "Inventing requirements from software: An empirical investigation with web services," in *RE'08*, 2008.

[31] E. Holbrook, J. Hayes, and A. Dekhtyar, "Toward automating requirements satisfaction assessment," in *RE'09*, 2009.