Analyzing the Impact of Natural Language Processing over Feature Location in Models

Raúl Lapeña
SVIT Research Group
Universidad San Jorge, Spain
rlapena@usj.es

Jaime Font
SVIT Research Group
Universidad San Jorge, Spain
jfont@usj.es

Óscar Pastor
Centro de Investigación en
Métodos de Producción de Software
Universitat Politècnica de València, Spain
opastor@pros.upv.es

Carlos Cetina
SVIT Research Group
Universidad San Jorge, Spain
ccetina@usj.es

Abstract

Feature Location (FL) is a common task in the Software Engineering field, specially in maintenance and evolution of software products. The results of FL depend in a great manner in the style in which Feature Descriptions and software artifacts are written. Therefore, Natural Language Processing (NLP) techniques are used to process them. Through this paper, we analyze the influence of the most common NLP techniques over FL in Conceptual Models through Latent Semantic Indexing, and the influence of human participation when embedding domain knowledge in the process. We evaluated the techniques in a real-world industrial case study in the rolling stocks domain.

CCS Concepts • **Software and its engineering** → *Model-driven software engineering*; **Reusability**; • **Information systems** → *Information retrieval*; • **Computing methodologies** → Natural language processing;

Keywords Feature Location, Natural Language Processing, Information Retrieval

ACM Reference Format:
Raúl Lapeña, Jaime Font, Óscar Pastor, and Carlos Cetina. 2017. Analyzing the Impact of Natural Language Processing over Feature Location in Models. In *Proceedings of 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3136040.3136052

GPCE'17, October 23–24, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5524-7/17/10…$15.00
https://doi.org/10.1145/3136040.3136052

1 Introduction

Feature Location is a common task in the Software Engineering (SE) field, specially in the maintenance and evolution of software products. The identification of Features helps with bug fixing, Feature reuse, or the formalization of Software Product Lines, amongst other tasks.

Feature Location techniques have been mainly applied to code. The results of Feature Location in code depend greatly in the style in which Feature Descriptions, code variables, and methods are written [17]. Therefore, Natural Language Processing (NLP) techniques are used to process them, since NLP has a direct and beneficial impact on the results.

There is a wide range of NLP techniques that can be used to process text prior to Feature Location [26]. Some of these techniques comprehend general phrase styling techniques (e.g.: lowercasing, removal of duplicate words), syntactical analysis techniques (filtering words through their role in a sentence, usually achieved through Parts-Of-Speech Tagging) [21], semantic analysis techniques (filtering of words according to their meaning, usually achieved through semantic root reduction of words to their lemmas) [29], and human-in-the-loop techniques (e.g.: domain terms extraction, stopwords extraction). These NLP techniques can be used independently of one another in most cases and scenarios, and are in fact combined in distinct ways by different authors depending on implementation circumstances and the particularities of their research.

For Feature Location in Conceptual Models, new techniques such as Formal Concept Analysis or Clustering have been emerging on the past years. However, the influence that NLP techniques (traditionally used for Feature Location in code) may have on Feature Location in Conceptual Models is a topic that has not received enough attention yet.

In the following pages, we present a set of custom UML Conceptual Models (described through the Background Section) from a real-world case study. The Models differ from the code, being the elements and relationships that conform them expressed with a distinct convention and manner than

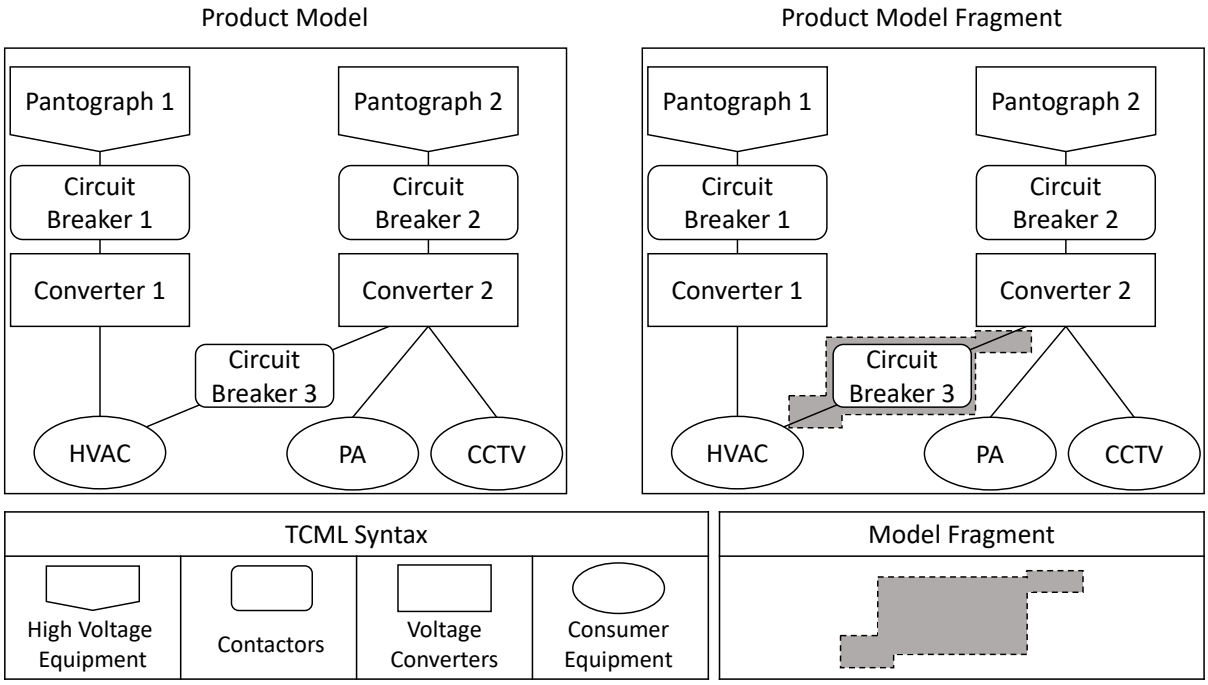

Figure 1. Example of Model and Model fragment

the ones typically used for coding. Therefore, it is not possible to assume without further validation that the NLP techniques that are used for Feature Location in code can be immediately extrapolated with success to Feature Location in Conceptual Models.

Through this work, we aim to analyze how the usage of different combinations of NLP techniques impacts Feature Location in Models through Latent Semantic Indexing (LSI) [19, 23], the technique that obtains the best Feature Location results [31]. Moreover, we elaborate on the impact of human participation over LSI results when embedding domain knowledge in the process, through the extended usage of stopwords and domain terms. We evaluated the techniques in a real-world industrial case study in the rolling stocks domain with our industrial partner, Construcciones y Auxiliar de Ferrocarriles (CAF).

Results show that using NLP techniques that achieve good results in Feature Location in code (such as Parts-Of-Speech Tagging or Lemmatizing) leads to a significant worsening of the rankings produced by LSI for Feature Location in Models. Through the Discussion of this work, we provide insight on why this is the case. For all the possible NLP combinations, embedding domain knowledge in the NLP process slightly improves the results. However, domain experts should decide whether participating in the NLP process to achieve this slight improvement is worth the effort and time involved.

The paper is structured as follows: Section 2 provides a background of the Models used through our work. Section 3 presents the Approach to our work. Section 4 formulates the Research Questions that we aim to respond through our work. Section 5 details the Experiment designed to tackle the Research Questions. Section 6 gives insight on the Discussion of the results of our work. Section 7 presents the Threats to Validity of our work. Section 8 summarizes the works related to the presented paper. Finally, Section 9 presents the conclusions of our work.

2 Background

This section presents the Domain Specific Language (DSL) used to formalize the Conceptual Models that implement the train control and management software of the products manufactured by our industrial partner, called Train Control Modeling Language (TCML). The TCML is a custom version of UML used by our industrial partner, that has the expressiveness required to describe both the interaction between the main pieces of equipment installed in a train unit, and the non-functional aspects related to regulation. It will be used through the rest of the paper to present a running example. For the sake of understandability and legibility, and due to intellectual property rights concerns, we present an equipment-focused simplified subset of the TCML.

Fig. 1 depicts one example, taken from a real-world train. It presents a converter assistance scenario where two separate pantographs (High Voltage Equipment) collect energy from the overhead wires, and send it to their respective circuit breakers (Contactors), which in turn send it to their independent Voltage Converters. The converters then power their assigned Consumer Equipment: the HVAC on the left (the

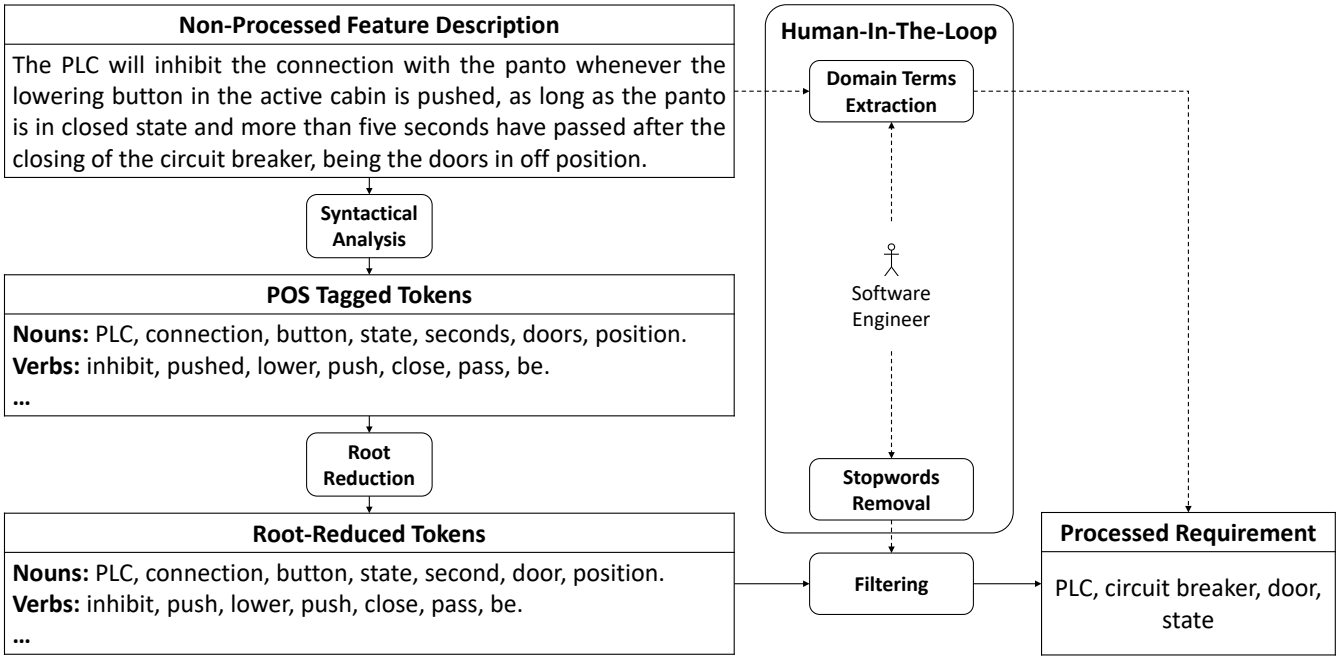

Figure 2. Compendium of NLP Techniques

train's air conditioning system), and the PA (public address system) and CCTV (television system) on the right.

To formalize the Model fragments used by through the rest of our work, we use the Common Variability Language (CVL) [18]. The elements of Fig. 1 highlighted in gray conform an example Model fragment, including one circuit breaker that connects Converter 2 to a Consumer Equipment assigned to Converter 1. This Model fragment is the realization of the "converter assistance" Feature, allowing the passing of current from one converter to equipment assigned to its peer for coverage in case of overload or failure of the first one.

3 Approach

So far, there has been no discussion on which NLP techniques should be applied, nor on which combination (or combinations) of NLP techniques yield the best results when used to carry out Feature Location in Models. Different authors use different techniques and combinations of techniques in their research, guiding their NLP efforts through implementation circumstances and problem particularities.

The goal of the presented approach is, by targeting a real-world industrial example, to study the impact of a set of combinations of the most spread NLP techniques over a widely accepted Feature Location process, Latent Semantic Indexing (LSI). By analyzing the success of LSI over different inputs, generated through the NLP of Feature Descriptions and Models by distinct technique combinations, we aim to determine which of them guide LSI to enhanced results.

The following subsections describe the NLP techniques taken in account through the rest of this work, and the combinations of techniques that are considered for further generation of LSI results.

3.1 NLP Techniques

Fig. 2 is used here to illustrate the whole compendium of techniques considered through this work, which are detailed one by one in this section, through the following paragraphs.

3.1.1 Baseline NLP

The most basic NLP technique covered in this work is the combination of tokenizing, lowercasing, and removal of duplicate keywords, which is often used as the most basic NLP technique for several widely-known LSI examples [14]. As such, we use it as the baseline for comparing the outcomes of the considered NLP techniques combinations. We disregard the complete lack of NLP as baseline since the scope of this work is the analysis of the outcomes of distinct NLP combinations of techniques, against verifying whether performing NLP over NL queries is better or not than not doing so.

3.1.2 Syntactical Analysis

Syntactical Analysis (SA) techniques split the words of NL sentences, analyzing the specific roles of each one of them in the sentence and determining their grammatical load. In other words, these techniques determine the grammatical function of each word in a particular sentence (e.g.: nouns,

verbs, adjectives, adverbs, etc.). These techniques, often referred to as Parts-Of-Speech Tagging (POS Tagging) techniques allow engineers to implement filters for words that fulfill specific grammatical roles in a sentence, usually opting for nouns, since these words are the ones that carry the relevant information about descriptions of Features and actions [4]. Words like verbs, adverbs, and adjectives are often filtered out and disregarded.

In Fig. 2, it is possible to appreciate the SA process, with the POS Tagged Tokens as outcome of syntactically analyzing a real-world NL Feature Description. Nouns and verbs are depicted while, for space reasons, the rest of the words are omitted in the Figure.

3.1.3 Root Reduction

Through the usage of semantic techniques such as Lemmatizing, words can be reduced to their semantic roots or lemmas. Thanks to lemmas, the language of the NL Feature Descriptions is unified, avoiding verb tenses, noun plurals, and strange word forms that interfere negatively with the Feature Location processes. Prior to carrying out Root Reduction (RR) techniques, it is imperative to use SA techniques, due to the fact that RR techniques are based on word dictionaries that are built upon the grammatical role of words in a sentence. The unification of the language semantics is an evolution over pure syntactical role filtering that allows for a more advanced filtering of words in NL Feature Descriptions.

In Fig. 2, it is possible to appreciate the RR process, with the Root-Reduced Tokens as outcome of the semantic analysis of the POS Tags derived from the NL Feature Description. For space reasons, only the lemmas of nouns and verbs are depicted once again.

3.1.4 Human-in-the-Loop

The inclusion of domain experts and, in particular, software engineers in Feature Location processes is a widely discussed topic within the SE community. It is often regarded as beneficial to have some sort of domain knowledge embedded in automated Feature Location systems, particularly on areas related to software reuse and software variability. Some of the techniques derived from humans interacting with Feature Location processes are Domain Terms Extraction and Stopwords Removal.

In order to carry out these techniques, Software Engineers provide two separate lists of terms: one list of terms (both single-word terms and multiple-word terms) that belongs to the domain and that must be always kept for analysis, and a list of irrelevant words that can appear throughout the entirety of the specification documents and that have no value whatsoever for the analysis. Both kinds of terms can be automatically filtered in or out of the final query, depending on the needs of the domain experts.

In Fig. 2, it is possible to appreciate the Human-in-the-Loop process, where a software engineer provides both lists of terms, which are consequently introduced into the final query, or filtered out of it. We do not consider separating the techniques and including only one of the lists. Should we include domain knowledge in our NLP, we should benefit of the whole of it and not take out a part.

3.1.5 Other Filters

Many other filters can be implemented to make a NL query (in our particular case, a Feature Description written in Natural Language) suit the needs of developers and researchers alike. For instance, a technique not covered in this study is Stemming (which consists in the reduction of words to their logical root or stem through a set of logically-related grammatical rules). We opted for the inclusion of Lemmatizing on its place, due to its more precise and advanced nature on the fulfillment of the same task [3].

3.2 NLP Configurations

When considering the possible configurations for the NLP techniques, we have taken in account the following facts, extracted from the prior subsection:

1. Root Reduction cannot be carried out without applying prior Syntactical Analysis.
2. We include or exclude domain knowledge as a whole.
3. The main goal of our work is to compare the distinct NLP techniques configurations, but we also aim to analyze the raw impact of embedding domain knowledge on NLP over Feature Location in Models.

Taking these three rules as our main standpoint, we have designed the table presented in Fig. 3. The Figure presents tree tiers of processing (baseline processing, processing with syntactical analysis, and full processing), split in two subgroups (excluding domain knowledge, and including domain knowledge), for a total of six possible configurations:

1. Baseline Processing (**BP**).
2. Baseline Processing + Domain Knowledge (**BP-DK**).
3. Tier 1 Processing (**T1**).
4. Tier 1 Processing + Domain Knowledge (**T1-DK**).
5. Tier 2 Processing (**T2**).
6. Tier 2 Processing + Domain Knowledge (**T2-DK**).

In order to test the impact of the aforementioned configurations over the Feature Location process, we apply them to process the input of a widely used Feature Location technique, Latent Semantic Indexing (LSI). We chose LSI due to it being the technique that offers best results in Feature Location [31].

3.3 Latent Semantic Indexing

LSI is an automatic mathematical/statistical technique that analyzes relationships between *queries* and *documents* (bodies of text). It constructs vector representations of both a user *query* and a corpus of text *documents* by encoding them as a

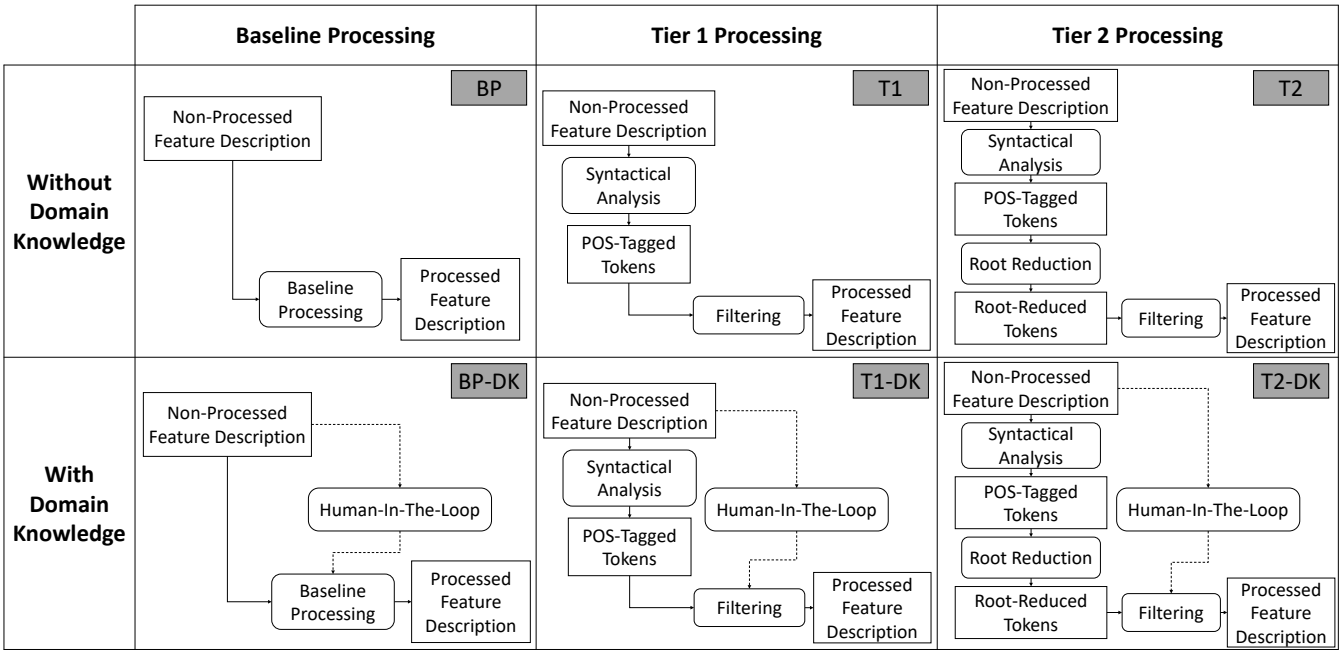

Figure 3. Possible NLP Techniques Configurations

term-by-document co-occurrence matrix, and analyzes the relationships between those vectors to get a similarity ranking between the *query* and the *documents*. Through LSI, we are able to extract a ranking of the Model fragments according to their similarity to a provided query Feature Description.

First, a textual representation of the Model fragments is obtained through the concatenation of the words that appear on its elements, and preprocessed with the same techniques as those used to preprocess the Feature Description. As an example, the full Model presented in the left part of Figure 1, after preprocessing, would yield the textual representation "Pantograph Circuit Breaker Converter HVAC Pantograph Circuit Breaker Converter Circuit Breaker PA CCTV".

Then, the *term-by-document co-occurrence matrix* is built. *Terms* are the words that compose the preprocessed Feature Description and the preprocessed textual representation of Model fragments. *Documents* are the preprocessed textual representations of the Model fragments. Finally, the *query* is formed by one processed Feature Description. Values of *term* occurrences in both the documents and the query are counted, and used to build the *term-by-document co-occurrence matrix*. The resulting *documents* and *query* columns are then transformed into vectors, and the relationships between the *documents* and the *query* are analyzed to extract a ranking of the most similar Model fragments for the Feature Description.

Figure 4 shows an example *term-by-document co-occurrence matrix*, with values associated to our case study, the vectors, and the resulting ranking. In the following paragraphs, an overview of the elements of the matrix is provided:

- Each row in the matrix stands for each unique keyword (*term*) extracted in the first step of our approach. In Figure 4, it is possible to appreciate a set of representative keywords in the domain such as 'pantograph' or 'doors' as the *terms* of each row.
- Each column in the matrix stands for the preprocessed text of each Model fragment in our case study. In Figure 4, it is possible to appreciate the identifiers of the Model fragments in the columns such as 'M_KAO001' or 'M_CIN072', representing the preprocessed text of those Model fragments.
- The final column stands for the *query*. In our approach, the *query* column stands for the preprocessed text of a Feature Description in our case study. In Figure 4, the identifier of the Feature Description in the *query* column ('R_BUD010') represents its preprocessed text.
- Each cell in the matrix contains the frequency with which the *term* of its row appears in the *document* denoted by its column. For instance, in Figure 4, the *term* 'pantograph' appears twice in the 'M_KAO001' preprocessed text and once in the 'R_BUD010' preprocessed text.

We obtain vector representations of the *documents* and the *query* columns by normalizing and decomposing the *term-by-document co-occurrence matrix* using a matrix factorization technique called *singular value decomposition* (SVD) [19, 23]. SVD is a form of factor analysis, or more properly the mathematical generalization of which factor analysis is a special case. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One component

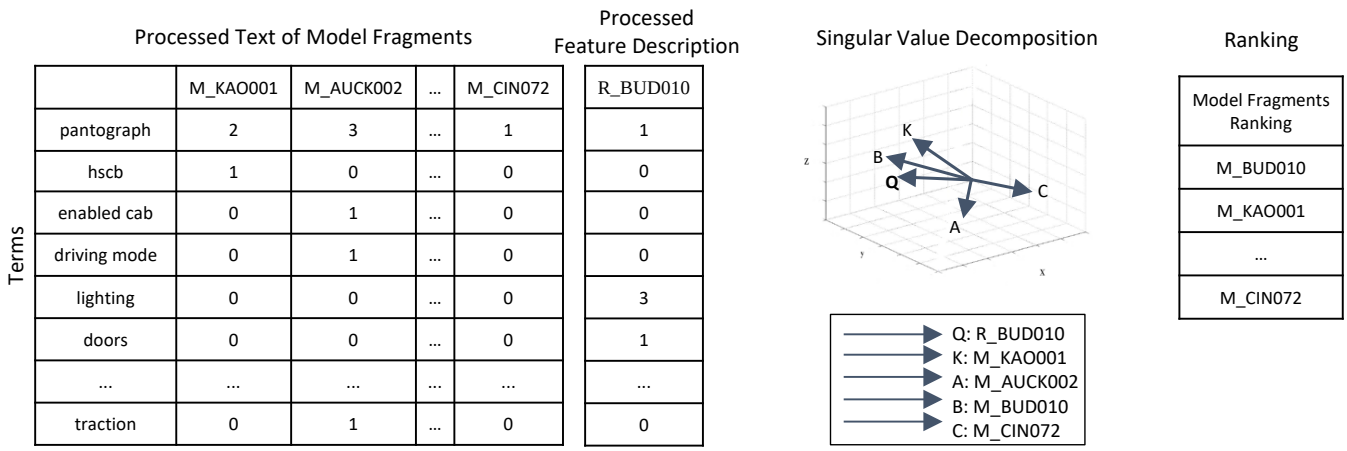

Figure 4. Latent Semantic Indexing Example

matrix describes the original row entities as vectors of derived orthogonal factor values, another describes the original column entities in the same way, and the third is a diagonal matrix containing scaling values such that when the three components are matrix-multiplied, the original matrix is reconstructed.

In Figure 4, a three-dimensional graph of the SVD is provided. On the graph, it is possible to appreciate the vectorial representations of some of the matrix columns. For space reasons, only a small set of the columns is represented. In the Figure, it is possible to appreciate the B vector ('M_BUD010' vector) as the closest to the Feature Description vector, followed by vectors K, C, and A, which are the vector representations of the columns highlighted in the left part of the matrix.

To measure the similarity degree between vectors, our approach calculates the cosine between the *query* vector and the *documents* vectors. Cosine values closer to one denote a higher degree of similarity, and cosine values closer to minus one denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more *terms* are shared between *documents*). Having this measurement, our approach orders the Model fragments according to their similarity degree to the Feature Description.

The relevancy ranking (which can be seen in Figure 4) is produced according to the calculated similarity degrees. In this example, LSI retrieves 'M_BUD010' and 'M_KAO001' in the first and second position of the product relevancy ranking due to the cosines being '0.9243' and '0.8454', implying a high similarity degree between the fragments and the Feature Description. On the other hand, 'M_CIN072' is returned in a latter position of the ranking due to its cosine being '-0.7836', a lower similarity degree.

4 Research Questions

Through this section, we aim to clearly establish what is the scope of our work, and to determine what are the research questions that we must tackle and have in mind when designing our comparison experiment. From the described problem, two research questions arise (RQ1 and RQ2), formulated in the following lines:

RQ1 How do different NLP configurations affect the efficiency and effectiveness of Feature Location in Models?

RQ2 How are human NLP efforts reflected in the outcome of Feature Location in Models?

Through the following section, we describe the experiment that we designed to address both research questions, as well as its results.

5 Experiment

Through the following subsections, we present our real-world case study, describe the oracle used for our experiment, detail the design of our experiment, and present the results of said experiment.

5.1 Case Study

We applied our experiment to a real-world case study from one of our industrial partners, CAF (Construcciones y Auxiliar de Ferrocarriles, available at `http://www.caf.net/en`). CAF is a worldwide provider of railway solutions. Their trains can be seen all over the world and in different forms (regular trains, subway, light rail, monorail, etc.).

A train unit is furnished with multiple pieces of equipment through its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their aim is to carry out specific tasks for the train. Some examples of these devices are traction equipment, compressors, brakes, the pantograph that harvests power from the overhead wires, etc. The control software of the train unit is

in charge of making all the equipment cooperate to achieve the train functionality, while guaranteeing compliance with the specific regulations of each country. The functionality of each train is detailed in documents where each desired Feature is specified through a Natural Language Description. In turn, the documents are implemented in Models.

For our experiment, CAF provided us with the Feature Descriptions and Models of five of their railway solutions, corresponding to the cities of Auckland, Bucharest, Cincinnati, Houston, and Kaohsiung. The trains are configured through about 100 Features per train, being each Feature described through one Feature Description. Feature Descriptions, in turn, have an average of 50 words. Regarding Models, each train is specified through no less than 8250 Model elements.

CAF also provided both a list of domain terms and a list of stopwords. The domain terms list comprehends around 300 domain terms, and the stopwords list comprehends around 60 words. Both lists were created by a domain expert from the company with a wide knowledge of the provided software products.

5.2 Oracle

In order to evaluate the results of our experiment, CAF provided us with their existing documentation on Feature Location between the Feature Descriptions in the specification documents and the Models. In said documentation on Feature Location, each Feature Description is mapped to a single Model fragment. A Model fragment is a subset of elements of a Model, specified with the Model fragment formalization capacities of the Common Variability Language (CVL) [18]. In other words, for each Feature, we know which is the associated Model fragment that implements it.

We use the mapping as the oracle for evaluating the impact of each NLP techniques configuration on LSI. In order to achieve this, we analyze the results of the rankings generated by LSI, checking the position of the ranking in which the oracle (correct Model fragment for the input Feature Description) appears.

5.3 Design of the Experiment

In order to tackle our research questions, we need to study:

1. How Feature Location techniques respond to the different NLP inputs.
2. Whether human-introduced NLP affects the process and if so, in which manner.

To that extent, we used the aforementioned configurations of NLP techniques along with the priorly described LSI. The steps through which our experiment is performed are detailed in the following paragraphs.

The first step is to select a NLP techniques configuration. With the chosen configuration, we perform the NLP of the text of all the Feature Descriptions in our case study. The same NLP is also applied to the textual representation of all the Model fragments in our case study.

From the NLP strings, all the resulting individual words are extracted to form a list of words. The list of words (*terms*), the NLP text of the Model fragments (*documents*), and the NLP text of one Feature Description (*query*) are used as input for the LSI technique.

LSI returns a ranking of Model fragments, ordered according to the similarity between their NLP textual representation to the NLP Feature Description. Through the ranking, and by leveraging the oracle knowledge, we can determine the ranking position in which the correct Model fragment for the provided Feature Description appears.

The LSI process is performed several times, taking each of the Feature Descriptions of our case study as query, in order to extract the NLP Model fragment rankings for all the available Feature Descriptions.

Analyzing the positions in which the correct Model fragments appear, we are able to evaluate the relative success or failure in terms of results and performance of the chosen NLP techniques configuration over the LSI process.

The described steps (choosing a NLP configuration, NLP of Feature Descriptions and Model fragments, LSI, impact analysis) are carried out for the six considered configurations. By comparing the results of the six configurations, we rate their global success or failure in terms of performance and impact on the resulting rankings, and analyze the statistical impact of humans on the NLP process.

In order to perform our experiment, we used a Lenovo E330 laptop, with an Intel® Core™ i5-3210M@2.5GHz processor, with 16GB RAM and Windows 10 64-bit.

5.4 Statistical Analysis

To properly compare the six NLP configurations, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [1].

In order to answer the RQs we perform statistical analysis to: (1) provide formal and quantitative evidence (statistical significance) that the six NLP configurations do in fact have an impact on the comparison metrics (i.e., that the differences in the results were not obtained by mere chance); and (2) show that those differences are significant in practice (effect size).

5.4.1 Statistical Significance

To enable statistical analysis, all of the configurations should be run a large enough number of times (in an independent way) to collect information on the probability distribution for each configuration. A statistical test should then be run to assess whether there is enough empirical evidence to claim (with a high level of confidence) that there is a difference between the two configurations (e.g., A is better than B). In order to do this, two hypothesis, the null hypothesis H_0 and the alternative hypothesis H_1, are defined. The null

hypothesis H_0 is typically defined to state that there is no difference among the configurations, whereas the alternative hypotheses H_1 states that at least one configuration differs from another. In such a case, a statistical test aims to verify whether the null hypothesis H_0 should be rejected.

The statistical tests provide a probability value, $p - value$. The $p - value$ receives values ranging between 0 and 1. The lower the $p - value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p - value$ under 0.05 is statistically significant [1], and so the hypothesis H_0 can be considered false.

The test that we must follow depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data; however, the Quade test is more powerful than the rest when working with real data [15]. In addition, according to Conover [5], the Quade test has shown better results than the others when the number of algorithms is low, (no more than 4 or 5 algorithms).

However, with the Quade test, is not possible to answer the following question: *Which of the configurations gives the best performance?* In this case, the performance of each configuration should be individually compared against all other alternatives. In order to do this, we perform an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each configuration, determining whether statistically significant differences exist among the results of a specific pair of configurations. In particular, we apply the Holm Post Hoc procedure, as suggested by Garcia et al. [15].

5.4.2 Effect Size

When comparing configurations with a large enough number of runs, statistically significant differences can be obtained even if they are so small as to be of no practical value [1]. Then, it is important to assess if a configuration performs statistically better than another and to asses the magnitude of the improvement. *Effect size* measures are used to analyze this.

For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [16, 37]. $\hat{A}_{12}$ measures the probability that using one configuration yields higher performance values than using another configuration. If the two configurations are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results 70% of the times with the first of the two configurations compared, and $\hat{A}_{12} = 0.4$ means that we would obtain better results 60% of the times with the second of the two configurations. Thus, we have an $\hat{A}_{12}$ value for every pair of configurations.

5.5 Results

For each NLP techniques configuration, we measured the best and worst position of the oracles in the rankings generated by LSI through the steps described in the previous section, as well as the average position in which the oracle appeared. In other words, for each configuration, we measured the best, worst, and average position of the correct Model fragment in the 500 rankings generated via LSI (one ranking per available Feature Description). We also measured the time that the execution of NLP took for the different configurations (average time of 25 executions). In the table, we do not highlight the LSI execution time averages, since it is practically identical for all the configurations (around 70 seconds). The amount of time associated to create the Domain Terms and Stopwords List artifacts used to embed the Domain Knowledge along the techniques (amounting up to around 3 hours) is also neglected in the table, being a one-time event that does not account for the performance of the techniques in the long term. Table 1 shows the results achieved by LSI when performed over the six configurations:

Table 1. Results per NLP techniques configuration

	Best Result	Worst Result	Average Result	Time Taken (s)
BP	#1	#14	#5	75
BP-DK	#1	#9	#4	79
T1	#11	#26	#16	287
T1-DK	#8	#21	#13	295
T2	#5	#19	#11	342
T2-DK	#3	#12	#9	376

In the table, it is possible to appreciate that the baseline processing leads LSI to the best results, with the baseline processing with embedded domain knowledge (BP-DK) achieving slightly better results than its fully automated counterpart (BP). Both baseline processing combinations achieve a ranking position #1 as their best result, with BP-DK achieving a ranking position #9 as its worst result (improving BP's #14 in 5 positions) and improving the average result of BP by 1 position. Regarding timing, BP improves BP-DK by 4 seconds on average after 25 executions of both.

From the results, it is also possible to discern that tier one processing, in both of its forms (T1 and T1-DK), leads LSI to worse results with regards to both ranking positions and performance. The best result presented by T1 is position #11, and the worst one, #26, with the average appearance of the oracle in position #16. T1-DK improves the results of T1 in 3 positions for the best result, 5 positions for the worst result, and 3 positions on average, but its results are still nowhere near those presented by the baseline. Regarding timing, the processing performed by T1 takes 287 seconds on average after 25 executions, beating T1-DK by 8 seconds, but surpassing the best timing result (BP's) by 212 seconds.

Finally, the table shows the results obtained by LSI after applying tier two processing to Feature Descriptions and

Model fragments. T2 obtains a #5 ranking position as its best result, a #19 ranking position as its worst result, and a ranking position of #11 on average. Embedding domain knowledge on tier two processing, once again, slightly improves the results of its fully automated counterpart. T2-DK obtains #3 as its best ranking position, #12 as its worst ranking position, and an average #9 position of the oracles in the rankings. From this results, it is observable that the results of tier two processing regarding ranking positions beat those of tier one processing, but are still far from those of the baseline. In addition, notice that both tier two combinations are utterly outperformed by the baseline combinations and the tier one combinations, since after 25 executions, T2 took 342 seconds and T2-DK took 376 seconds on average, which is 267 and 301 seconds slower than the best average time, respectively.

5.5.1 Results Statistics

The Quade test applied to the results provides $p - Values$ smaller than 0.05, which reject the null hypothesis for all the configuration pairs. Consequently, we can state that there exist differences in the configurations for the performance indicator evaluated (mean position in the ranking).

Table 2. Holm's post hoc $p - Values$ and $\hat{A}_{12}$ statistic for each pair of configurations

Configurations	$p - values$	$\hat{A}_{12}$ measures
BP-DK vs BP	$4.4x10^{-5}$	0.5966
T1 vs BP	$\ll 2x10^{-16}$	0.0188
T1-DK vs BP	$\ll 2x10^{-16}$	0.0711
T2 vs BP	$\ll 2x10^{-16}$	0.1572
T2-DK vs BP	$\ll 2x10^{-16}$	0.2605
T1 vs BP-DK	$\ll 2x10^{-16}$	0
T1-DK vs BP-DK	$\ll 2x10^{-16}$	0.0183
T2 vs BP-DK	$\ll 2x10^{-16}$	0.0818
T2-DK vs BP-DK	$\ll 2x10^{-16}$	0.1628
T1-DK vs T1	$4.1x10^{-15}$	0.7082
T2 vs T1	$\ll 2x10^{-16}$	0.7962
T2-DK vs T1	$\ll 2x10^{-16}$	0.9319
T2 vs T1-DK	$4.2x10^{-13}$	0.6283
T2-DK vs T1-DK	$\ll 2x10^{-16}$	0.7976
T2-DK vs T2	$\ll 2x10^{-16}$	0.6669

Table 2 shows the results of the statistical analysis performed (statistical significance and effect size). The first column shows each pair of configurations, the second column shows the $p - Values$ of Holm's post hoc analysis for each pair of algorithms, and the third column shows the $\hat{A}_{12}$ statistic for each pair of configurations.

All the $p - Values$ shown in this table are smaller than their corresponding significance threshold value (0.05), indicating that the differences of performance between those configurations are significant.

Regarding the effect size, the largest differences were obtained between the T1 and BP-DK configurations (where the BP-DK configuration achieves better results than the T1 configuration 100% of times). In general, BP and BP-DK configurations obtain better results than the competitors for all the cases. When comparing BP and BP-DK, the configuration that includes Domain Knowledge (BP-DK) will produce better results around 60% of times.

6 Discussion

The results presented in the previous section suggest that, when faced with Feature Location in Models, the baseline processing with embedded domain knowledge guides LSI to achieving the best possible results. The usage of more advanced techniques, on the other hand, leads to worse retrieval of results. Analyzing the Feature Descriptions, the Models, and the overall process, we noticed a series of facts that help explain why this is the case:

1. Verbs and adjectives do appear in the Models, and thus hold a vital amount of information for the Feature Location process. In addition, these words do not vary amongst Feature Descriptions and Models. For instance, verbs tend to appear always in the infinitive form (raise, open, etc.), and adjectives are invariable (electric, empty, etc.). This is not the case for nouns, which can and in fact do appear in different forms. For instance, singular and plural forms are indistinctly used through the Feature Descriptions (door vs. doors), while in the Models, it is extremely rare to find a plural form of a noun (there are only 2 occurrences of plural noun forms in the Models of the 5 trains), since multiplicity is defined through relationships between elements and not expressed in the textual representation of Models.
2. Our approach first uses POS Tagging to identify the tags of the words, and then uses said tags to filter every word out of the NLP process, except for nouns. This fact is propagated to Lemmatizing, which relies on the outcome of the filtering to perform the necessary operations to obtain the lemmas of the words. Due to the prior fact (verbs and adjectives do hold relevant information in the Models), our usage of these more complex processes removes a portion of the available information of the problem, which is useful for the Feature Location process, while the baseline does not cause this phenomena. This explains the fact that both the tier one processing and the tier 2 processing lead

LSI to worse results than the baseline. In the future, we may consider adding other tags such as verbs and adjectives to the list of words that are not to be filtered out, checking whether their inclusion can lead to Feature Location improvements.

3 As stated before, the noun words that appear in the Feature Descriptions are used in different forms in an indiscriminate manner. When performing the tier one processing, using only POS Tagging, this leads to a worsening of the results. The usage of tier two processing includes Lemmatizing, which serves as a bridge that unifies the language between the Feature Descriptions and the Model parts that are left after POS Tagging. The unification of the language is what causes the improvement that can be appreciated from tier one results to tier two results. Still, since Lemmatizing is performed after POS Tagging, a huge part of the information is already lost, and thus, the provided results do not reach the levels of those provided by the baseline. Due to this fact, we may consider testing the baseline with the inclusion of Lemmatizing in the future, while ignoring the POS Tagging filtering. We believe that, since the baseline includes nouns, it can also benefit from the language unification provided by Lemmatizing.

4 Even if we may consider testing other combinations of techniques or improving the processes through which they are being used in this work in order to test the performance of results as a case study, time performance results discourage us to use these techniques for real-world SE tasks in the environment of the company we are working with. In contrast with our reduced dataset, where the operations can be addressed in a quick and easy manner, in a real-world environment thousands of Models and Feature Descriptions must be taken in account, and the execution time of the different techniques grows exponentially. Time performance is key for the competitiveness of a company such as our industrial partner, and thus software engineers tasked with Feature Location cannot afford to wait for results for as long as some NLP techniques require. Upon revisiting the results, software engineers immediately preferred the baseline processing, since it is easier to understand, implement, and manage, as well as quicker in its execution, allowing them to review Model fragments and retrieve Feature Location results faster, by post-processing the given rankings through their domain knowledge and expertise.

5 We confirmed that adverbs and other connector words do not appear in the Models. Therefore, when counting occurrences of those words in the Models, the result is always zero. This leads to the fact that, by default, a percentage of the rows in the LSI term-by-document co-occurrence matrix are introduced with no information and therefore, do not contribute to the solving of the problem. A considerable percentage of these words appear in the stopwords list provided by the software engineers. The removal of these words when processing the available texts, in turn, causes the removal of their irrelevant rows from the LSI matrix. Part of the improvement caused by humans on Feature Location results is due to this fact, specially in the case of the baseline, where the inclusion of domain terms does not play a part in the improvement that can be appreciated between BP and BP-DK (see the next point for more details on this).

6 In the case of the baseline, the inclusion of domain terms does not cause a great impact on the results. After all, when using the baseline processing, all the words are included in the LSI analysis. This is not the case for POS Tagging or Lemmatizing, where the inclusion of domain terms (which are often composed, containing adjectives) causes an inclusion of information that would otherwise be discarded. On the other hand, adverbs and other connector words are discarded by POS Tagging, so the inclusion of rows with no information is a phenomena that does not occur. Looking at statements **4** and **5** altogether, we can observe that, due to the behavior of the NLP techniques, we can only benefit from one specific aspect of the domain knowledge at a time, depending on which techniques we are leveraging to guide the Feature Location process (stopwords for baseline processing, domain terms for POS Tagging based techniques).

7 Nevertheless, the evidence suggests that even though human-introduced processing improves FL in all scenarios, its real impact is not significant for the results. Improving an average of 1 to 3 positions in a ranking of 500 Model fragments is not a real enhancement of FL. Domain knowledge should only be provided and embedded in NLP in cases where this is an almost immediate process. We do not recommend having software engineers employ time and effort in this task, but rather on more important, impactful duties.

These facts can be summarized as responses to our previously asked research questions:

RQ1 How do different NLP configurations affect the efficiency and effectiveness of Feature Location in Models?

The baseline processing yields the best results when used to guide Feature Location in Model fragments. It outperforms more complex techniques in both results and time. More complex techniques can lead to losses of information in this field, and their execution times render them impractical in real-world scenarios.

RQ2 How are human NLP efforts reflected in the outcome of Feature Location in Models?
Human NLP efforts improve the outcome of Feature Location in Model fragments in every chosen combination of techniques for different reasons, but the improvement is slight.

7 Threats to Validity

In this section, we use the classification of threats of validity of [32, 39] to acknowledge the limitations of our approach:

1. **Construct Validity:** This aspect of validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind. In order to minimize this risk, we study the positions of the oracles in the rankings, an objective and widely accepted measure, used before by other researchers in the community [17].
2. **Internal Validity:** This aspect of validity is of concern when causal relations are examined. There is a risk that the factor being investigated may be affected by other neglected factors. The number of Features and Models presented in this work may look small, but they implement a wide scope of different railway equipment.
3. **External Validity:** This aspect of validity is concerned with to what extent it is possible to generalize the finding, and to what extent the findings are of relevance for other cases. Both NL-expressed Features and Conceptual Models are frequently leveraged to specify all kinds of different software. LSI is a widely accepted and utilized technique which has proven to obtain good results in multiple domains. The NLP techniques studied through this work are also commonly used in the whole of the SE community. Therefore, our experiment does not rely on the particular conditions of our domain. Nevertheless, our findings are based on a single study. Therefore, the experiment and its results should be replicated with different kinds of models and in other domains before assuring their generalization.
4. **Reliability:** This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. The Feature Descriptions and Models of the trains used through our experiment were provided by our industrial partner engineers, as well as the domain terms and stopwords lists, which were crafted by domain experts not involved in this research.

8 Related Work

The role of NLP is vital to the Software Engineering community [33]. NLP has been applied to tackle different issues in software engineering at several levels of abstraction. Works like [35, 36] or [7], among many others, use NLP to tackle specific problems and tasks, but do not study the implications of using different NLP techniques or combinations of techniques over the results, as our work does.

In [10], the authors use NLP techniques to identify equivalence between NL software artifacts. The authors also define and use a series of principles for evaluating the performance of NLP when identifying said equivalence. They conclude that, in their field, the performance of NLP is determined by the properties of the given dataset over which it is performed. They measure the properties as a factor to adjust the NLP process and performance accordingly, and then apply their principles to an industrial case study. Our work differs from [10], since the authors do not tackle the impact of different NLP configurations as we do, but rather adjust the NLP process according to a series of principles derived from dataset properties. Moreover, [10] studies equivalence between NL software artifacts, while we analyze NLP configurations to process NL software artifacts for Feature Location in Models. In addition, we do not define a set of principles to serve as guidelines on which NLP configuration to use, but rather present the results of applying the NLP configurations to our case study, exposing the way they behave and improve (or worsen) each other. Finally, we present an in-depth study on how human involvement in the NLP process affects Feature Location in Models, an issue that [10] does not tackle.

The work presented in [2] uses NLP to study how changes in NL software artifacts impact other artifacts of the same kind in the same specification. Through the pages of their work, the authors analyze the traceability links between NL software artifacts, and use NLP to determine how changes must propagate. Opposite to [2], our work does not analyze changes in NL software artifacts or how they affect the system. Instead, we put the focus on what is the most appropriate way of applying NLP to said artifacts for Feature Location in Models. Moreover, the authors of [2] do not consider different configurations for their NLP, but rather guide the process by taking in consideration the properties of the artifacts.

The work presented in [8] takes in consideration the possible configurations of LSI when using the technique for traceability links recovery between software artifacts, namely requirements and test cases. In their work, the authors state that the configurations of LSI depend on the datasets used, and they look forward to automatically determining an appropriate configuration for LSI for any given dataset. In our work, we do not tackle different LSI configurations or how LSI configurations impact the results of traceability recovery between requirements and Models, but rather analyze how different NLP configurations affect the results of Feature Location in Models.

Other approaches related to the Feature Location process presented in this paper comprehend Feature and Requirement location techniques. Through the following paragraphs, we discuss said approaches and compare our work to them.

Typechef [22] provides an infrastructure to locate the code associated to a given Feature by means of analyzing the #ifdef directives. Trace analysis [9] is a run-time technique used to locate Features. When the technique is executed, it produces traces indicating which parts of code have been executed. Some approaches related to Feature location use LSI to extract the code associated to a Feature [25, 30]. These techniques have been generally applied to search the code of a Feature in a given individual product. The main goal of our approach, in contrast, is to analyze how NLP configurations impact Feature Location in Models.

Feature location approaches in a product family such as the one presented in [40] center their efforts in finding the code that implements a Feature between the different products by combining techniques such as FCA [13] and LSI. In our approach, we are not interested in the code representation of a Feature in the family, but in finding the NLP configuration that guides LSI to better results when used for Feature Location in Models.

Other works such as [34] focus on applying reverse engineering to the source code to obtain the variability Model. In [6] the authors use propositional logic which describes the dependencies between Features. In [28] the authors combine Typechef techniques and propositional logic to extract conditions among a collection of Features. These works engage explicitly the variability of products, but do not tackle NLP configurations and their impact on Feature Location in Models, as our work does.

In [24], Lapeña et al. use POS Tagging in combination with an adapted two-step LSI to obtain rankings of methods for all the requirements of a new product in a product family. The scope of the presented work, on the other hand, is centered around analyzing how distinct NLP configurations affect the results of Feature Location in Models.

Some works [20, 27, 38, 41, 42] focus on Feature Location in Models by comparing the Models with each other to formalize the variability among them in the form of a Software Product Line. The presented work differs from these works in that the aim is not to formalize the variability, but to analyze the impact that NLP configurations have on Feature Location in Models.

Finally, Font et al. [11] use a Single Objective Evolutive Algorithm (SOEA) to locate Features among a family of Models in the form of a variation point. Their approach is refined in [12], where the authors use a SOEA to find sets of suitable Feature realizations. The authors first cluster Model fragments based on their common attributes into Feature realization candidates through Formal Concept Analysis, and then Latent Semantic Indexing ranks the candidates based on the similarity with the Feature description. The presented approach, in contrast, analyzes how NLP configurations affect Feature Location in Models.

9 Conclusions

Natural Language Processing (NLP) techniques have been extensively used to preprocess the language of software artifacts for Feature Location in code, due to the direct and positive impact they have on the outcome. However, the impact that these techniques have on the results of Feature Location in Models is an issue that has not been tackled yet.

Through this paper, we analyze how NLP techniques affect the outcome of Feature Location in Models. We process the Feature Descriptions and Models from a real-world industrial case study through combinations of NLP techniques, and perform Latent Semantic Indexing (LSI) over the processed specifications. We study the rankings produced by LSI with our oracle to evaluate the impact and repercussions of the NLP techniques over the Feature Location process.

Results show that using NLP techniques that have achieved good results in the past for Feature Location in code leads to a significant worsening of the rankings in the case of Feature Location in Models when compared to a Baseline Processing. We were able to identify a series of issues that cause this effect. In addition, our results highlight that embedding domain knowledge in the NLP process improves the Feature Location results, although in a non-significant manner. Domain experts should decide whether participating in the NLP process is worth the effort and time involved. The findings of our work are useful since:

1. Thanks to the retrieved results, we found out that we should not get carried away by inertia and apply NLP as we do in Feature Location in code. Advanced NLP techniques do improve the Feature Location results in the code realm, but we cannot assume that they will do so in the Models field as well. In fact, using these techniques by inertia may lead us to a worsening of the results.
2. Regarding domain experts participation in the NLP process, our results shed light on their true impact over Feature Location in Models. Thanks to that, domain experts can better value if the time and effort inverted in participating in the NLP process does pay off in terms of results improvement.
3. Finally, the Discussion of this work about NLP techniques configurations identifies specific issues that must be tackled in order to apply NLP in domains where BP-DK does not guide Feature Location in Models to proficient results.

Acknowledgments

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R) and ITEA3 15010 REVaMP2 Project. We also thank Fundación Banco Sabadell.

References

[1] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Softw. Test. Verif. Reliab.* 24, 3 (May 2014), 219–250. https://doi.org/10.1002/stvr.1486

[2] Chetan Arora, Mehrdad Sabetzadeh, Arda Goknil, Lionel C Briand, and Frank Zimmer. 2015. Change impact analysis for natural language requirements: An NLP approach. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*. IEEE, 6–15.

[3] Vimala Balakrishnan and Ethel Lloyd-Yemoh. 2014. Stemming and lemmatization: a comparison of retrieval performances. *Lecture Notes on Software Engineering* 2, 3 (2014), 262.

[4] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. 2009. On the role of the nouns in IR-based traceability recovery. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 148–157.

[5] W. J Conover. 1999. *Practical Nonparametric Statistics, 3rd Edition*. Wiley.

[6] Krzysztof Czarnecki and Andrzej Wasowski. 2007. Feature Diagrams and Logics: There and Back Again. In *Proceedings of the 11th International Software Product Lines Conference*.

[7] Chuan Duan and Jane Cleland-Huang. 2007. Clustering support for automated tracing. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 244–253.

[8] Sebastian Eder, Henning Femmer, Benedikt Hauptmann, and Maximilian Junker. 2015. Configuring latent semantic indexing for requirements tracing. In *Proceedings of the Second International Workshop on Requirements Engineering and Testing*. IEEE Press, 27–33.

[9] Andrew David Eisenberg and Kris De Volder. 2005. Dynamic Feature Traces: Finding Features in Unfamiliar Code. In *21st IEEE International Conference on Software Maintenance*.

[10] Davide Falessi, Giovanni Cantone, and Gerardo Canfora. 2013. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering* 39, 1 (2013), 18–44.

[11] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm. In *Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness*.

[12] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*.

[13] Bernhard Ganter and Rudolf Wille. 2012. *Formal Concept Analysis: Mathematical Foundations*. Springer Science & Business Media.

[14] Edel Garcia. 2006. Latent Semantic Indexing (LSI) A Fast Track Tutorial. *Grossman and Frieders Information Retrieval, Algorithms and Heuristics, 2006* (2006).

[15] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. 2010. Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. *Inf. Sci.* 180, 10 (May 2010), 2044–2064. https://doi.org/10.1016/j.ins.2009.12.010

[16] R. J. Grissom and J. J. Kim. 2005. *"Effect sizes for research: A broad practical approach.* Mahwah, NJ: Earlbaum.

[17] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 842–851.

[18] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K Olsen, and Andreas Svendsen. 2008. Adding standardized variability to domain specific languages. In *Software Product Line Conference, 2008. SPLC'08. 12th International*. IEEE, 139–148.

[19] Thomas Hofmann. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 50–57.

[20] Sönke Holthusen, David Wille, Christoph Legat, Simon Beddig, Ina Schaefer, and Birgit Vogel-Heuser. 2014. Family Model Mining for Function Block Diagrams in Automation Software. In *18th International Software Product Lines Conference*.

[21] Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 216–223.

[22] Christian Kästner, Paolo G. Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. 2011. Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

[23] Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse processes* 25, 2-3 (1998), 259–284.

[24] Raúl Lapeña, Manuel Ballarín, and Carlos Cetina. 2016. Towards Clone-and-Own Support: Locating Relevant Methods in Legacy Products. In *Proceedings of the 20th International Conference on Software Product Lines*.

[25] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. 2007. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*.

[26] Christopher D Manning, Hinrich Schütze, et al. 1999. *Foundations of statistical natural language processing*. Vol. 999. MIT Press.

[27] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Bottom-up Adoption of Software Product Lines: a Generic and Extensible Approach. In *Proceedings of the 19th International Conference on Software Product Lines*.

[28] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. 2014. Mining Configuration Constraints: Static Analyses and Empirical Results. In *36th International Conference on Software Engineering*.

[29] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. 2004. A rule based approach to word lemmatization. In *Proceedings C of the 7th International Multi-Conference Information Society IS 2004*, Vol. 1. Citeseer, 83–86.

[30] Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Václav Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Trans. Software Eng.* 33, 6 (2007).

[31] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. 2007. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering* 33, 6 (2007).

[32] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

[33] Kevin Ryan. 1993. The role of natural language in requirements engineering. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. IEEE, 240–242.

[34] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wasowski, and Krzysztof Czarnecki. 2011. Reverse Engineering Feature Models. In *Proceedings of the 33rd International Conference on Software Engineering*.

[35] Hakim Sultanov and Jane Huffman Hayes. 2010. Application of swarm techniques to requirements engineering: Requirements tracing. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*. IEEE, 211–220.

[36] Senthil Karthikeyan Sundaram, Jane Huffman Hayes, Alex Dekhtyar, and E Ashlee Holbrook. 2010. Assessing traceability of software engineering artifacts. *Requirements engineering* 15, 3 (2010), 313–335.

[37] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101 arXiv:http://jeb.sagepub.com/content/25/2/101.full.pdf+html

[38] David Wille, Sönke Holthusen, Sandro Schulze, and Ina Schaefer. 2013. Interface Variability in Family Model Mining. In *17th International Software Product Line Conference*.

[39] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[40] Yinxing Xue, Zhenchang Xing, and Stan Jarzabek. 2012. Feature Location in a Collection of Product Variants. In *19th Working Conference on Reverse Engineering*.

[41] Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. 2011. Model Comparison to Synthesize a Model-Driven Software Product Line. In *Proceedings of the 15th International Conference on Software Product Lines*.

[42] Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. 2012. Augmenting Product Lines. In *19th Asia-Pacific Software Engineering Conference*.