Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.DOI

Evaluating Low-cost in Internal Crowdsourcing for Software Engineering: The Case of Feature Location in an Industrial Environment

Francisca Pérez, Ana C. Marcén, Raúl Lapeña, Carlos Cetina
SVIT Research Group. Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain
{mfperez, acmarcen, rlapena, ccetina}@usj.es

Corresponding author: Francisca Pérez (e-mail: mfperez@usj.es).

ABSTRACT

Internal crowdsourcing in software engineering is a mechanism for recruiting engineers to carry out more efficiently software engineering tasks. However, engineers are busy resources and time is a valuable asset in industry, which hinders internal crowdsourcing in software engineering from becoming a widespread practice. In this work, we propose a low-cost variant of internal crowdsourcing for locating features in models, which limits the time that engineers can spend for providing knowledge. Our approach uses the knowledge provided by the internal crowd to automatically reformulate an initial feature description. The result is taken as input to automatically locate the relevant model fragment using Latent Semantic Indexing. We evaluate our approach using four query reformulation techniques in a real-world case study from our industrial partner. We compare the results of our approach in terms of recall, precision and F-measure with a baseline by means of statistical methods to show that the impact of the results of our approach is significant. Despite the limitation of time, the results show that low-cost in internal crowdsourcing improves significantly the results in an industrial context where engineers' availability is scarce.

INDEX TERMS Crowdsourced Software Engineering, Crowdsourcing, Collaborative Information Retrieval, Query Reformulation, Feature Location

I. INTRODUCTION

Crowdsourcing in software engineering [1] uses an open call format to recruit software engineers to cooperate in carrying out various types of software engineering tasks such as requirements extraction, design, coding and testing. Since crowdsourcing in software engineering has been claimed to lower costs and defect rates, it has increasing interest [1].

Studies show that software engineers spend about 85% of the total effort in software maintenance and evolution [2]. Feature Location (FL), one of the most important tasks undertaken during software maintenance [3], is the process of finding the set of software artifacts that realize a specific functionality. Most of the existing works in the literature perform FL on code as the software artifact [4], [5], whereas model artifacts have been neglected. To achieve FL, software engineers often use search engines that need a query as input, which describes the target feature in Natural Language (NL). Unfortunately, the engineers generally lack the idea of the software artifacts that realize a target software feature in an industrial context where software has been developed over years by different engineers. Hence, queries hardly lead to relevant results, so the queries need to be reformulated (e.g., to add more appropriate terms). Despite the engineers' issues for locating features and the crowdsourcing benefits, crowdsourcing is not a widespread practice for FL.

Previous works [6]–[9] use external crowdsourced knowledge (Stack Overflow) to reformulate queries for code search. However, external crowdsourced knowledge cannot provide relevant information for specific industrial contexts. Therefore, internal crowdsourced knowledge provided by software engineers of the company is needed for feature location. However, engineers are busy resources and time is a valuable

asset in industry. This hinders engineers' willingness to invest in internal crowdsourcing.

To cope with the lack of internal crowdsourcing approaches for locating features in models, the contribution of this paper is twofold.

1) We propose a novel approach that is a low-cost variant of internal crowdsourcing in software engineering for performing FL in models. The presented approach limits the time that engineers spend for providing knowledge (a description in NL of the target feature and a self-rated level of confidence). Then, the approach leverages the provided knowledge to automatically reformulate an initial feature description. The reformulated feature description is used as the input query for Latent Semantic Indexing (LSI) [10], which automatically locates the most relevant model fragment for the reformulated feature description. We select LSI since it is used in previous studies that locate features in code [4], [5] and it is the technique that obtains the best results for FL tasks [11]–[13].
2) We evaluate our approach in a real-world case study through four existing query reformulation techniques, three expansion techniques (Rocchio, RSV, and Dice) and query reduction. The evaluation is performed by applying the techniques to a real-world case study provided by our industrial partner (*Construcciones y Auxiliar de Ferrocarriles*, CAF)[1], a world reference in train manufacturing. CAF provided us with the models of the software that controls and manages the trains, 43 feature names to be located, and an oracle (the ground truth, comprising the model fragments that materialize the target features). Moreover, 19 domain experts from CAF provide internal crowdsourced knowledge. We compare the model fragment from the oracle with the model fragment from our approach to measure performance through recall, precision, and F-measure. In addition, we compare the results of our approach with a baseline by using statistical methods, in order to provide both formal and quantitative evidence of the impact of the obtained results, thus proving the statistical significance of our work.

Despite the limitation of time, the results show that low-cost in internal crowdsourcing improves significantly the results in an industrial context where engineers' availability is scarce. Our approach improves the F-measure of the baseline by 27.66%. The obtained results highlight the importance of internal crowdsourcing in industrial environments, and promote the adoption of internal crowdsourcing in different domains.

The rest of the paper is structured as follows: Section II presents an overview of our approach. Section III describes both how our approach homogenizes the NL feature descriptions and four automatic query reformulation techniques. Section IV describes how the relevant model fragment is obtained from the reformulated feature description. Section V presents the evaluation, and Section VI shows the results. Section VII describes the threats to validity. Section VIII reviews the related work. Finally, Section IX concludes the paper.

[1] www.caf.net/en

II. OVERVIEW OF THE APPROACH

Crowdsourced Software Engineering generally involves the following types of actors (or stakeholders) [1]: requesters and workers. Requesters have a software engineering task that needs to be done, whereas workers are recruited to work on the software engineering task.

Figure 1 shows the inputs that these actors provide in our proposed approach as well as an overview of the approach to support low-cost in internal crowdsourcing for FL in models. To start with, a requester provides a feature description from the name of the target feature to be located in models. The provided description is set as the input query.

Second, the feature name is also used to recruit workers who have been working in the company (i.e., internal workers). The recruitment of internal workers is done manually by the requester or by the company based on different criteria such as workers' availability and knowledge. Each recruited worker contributes to the internal crowdsourcing by providing a feature description of the target feature and a self-rated confidence level. The self-rated level of confidence is provided through the use of a Likert scale ranging from 7 (highest confidence) to 1 (lowest confidence). We choose 7-point Likert items because they have been shown to be more accurate, easier to use, and a better reflection of a respondent's true evaluation [14]. The approach limits the time that each worker has to provide the knowledge. This limitation aims to lower the costs of internal crowdsourcing since the time and availability of workers is scarce in industrial contexts. We set the time limit to 10 minutes per task. This decision is made based on the majority of responses from the workers of our industrial partner to the question of how much time they would be willing to invest in a task of internal crowdsourcing that was not going to significantly interrupt their work.

Third, internal crowdsourcing is supported by automatically reformulating the query (i.e., the feature description provided by the requester) with the feature descriptions provided by the workers. Instead of automatically reformulating the query, workers can just discuss together and collaboratively to manually draft a better description of the query. However, this manual reformulation is time-consuming in complex projects, specially in industrial environments where a vast amount of software is accumulated over the years. This is because the combination of a set of feature descriptions in a single query requires much time since workers have to both understand all feature descriptions that have been provided and discuss about the most adequate terms to locate the target feature. In order to avoid this time-consuming manual reformulation, the combination of a set of feature descriptions in a single query is automatically performed

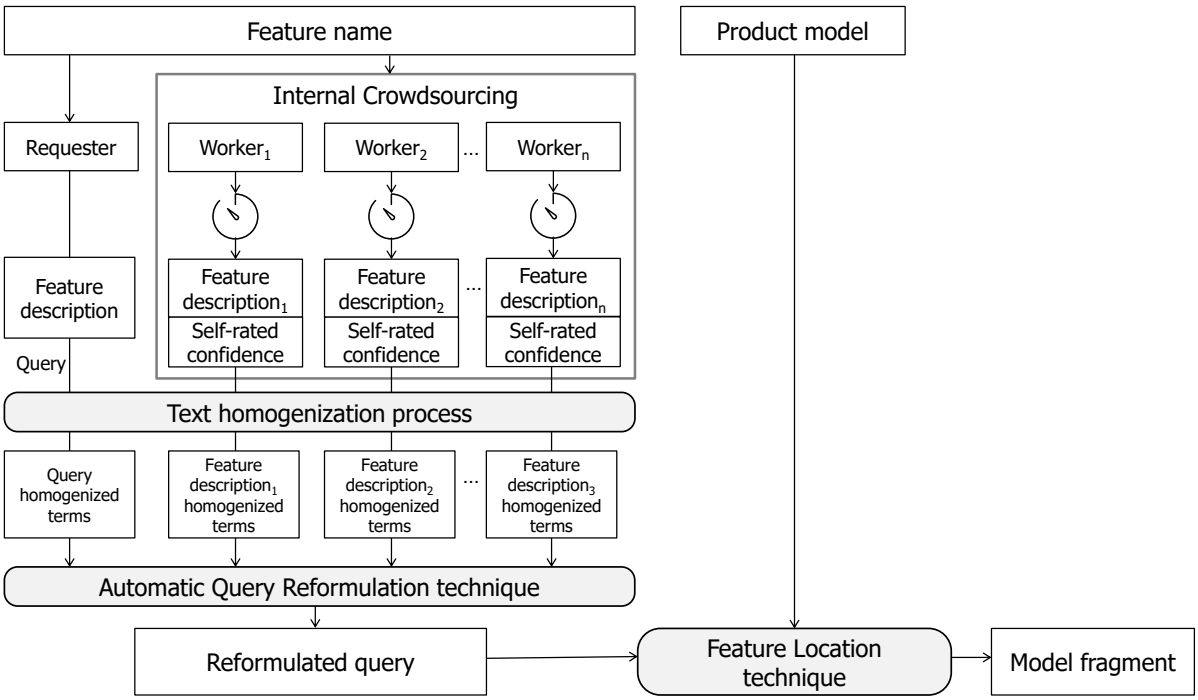


FIGURE 1: Overview of the approach

using automatic query reformulation techniques. Previous works have proposed several approaches to automatically reformulate a query, grouped into two categories [15], depending on whether they expand or reduce the query.

Finally, both the model and the reformulated query are used as input to locate the relevant model fragment through a FL technique based on Latent Semantic Indexing (LSI). We select LSI since it is the technique that obtains the best results for FL tasks [16].

The next two sections describe different reformulation techniques (three query expansion techniques and one query reduction technique) where each technique can be used as an alternative to support internal crowdsourcing in our approach, and how FL is performed using LSI.

III. SUPPORTING INTERNAL CROWDSOURCING

The goal of automatic query reformulation is to automatically define a new query, starting from an initial one, which is able to lead to improved retrieval results. To support internal crowdsourcing, our approach applies an automatic query reformulation technique. Over time, a large variety of automatic query reformulation techniques have been proposed that either expand the initial query or reduce it.

The following three subsections present: the homogenization process that our approach performs to the feature descriptions, three techniques of query expansion and one technique of query reduction. Each of these techniques can be used to obtain the reformulated query in our approach, so different variants of our approach can be executed. We selected these techniques because they perform best in the NL document retrieval field [17].

A. HOMOGENIZING THE TEXT OF FEATURE DESCRIPTIONS

Our approach homogenizes the NL text of the feature descriptions before the initial query is reformulated by combining Natural Language Processing (NLP) techniques. Homogenizing the text is often regarded as beneficial and a frequent practice that is performed using a combination of parts-of-speech tagging, removal of stopwords, and stemming [18]. We adopt this homogenization practice to process the feature descriptions in the following way:

1) First, the text is split into words (tokens). A white space tokenizer is typically good enough to split the text, but more complex tokenizers (such as a CameCase naming tokenizer) may need to be applied for some sources.
2) Secondly, Parts-of-Speech (POS) tagging is applied to analyze the grammatical roles of the words, inferring their roles in the provided text. The tagging of the words allows for the removal of words that not carry any relevant information (such as prepositions).
3) Then, stemming is applied to unify the language in use in the text. Stemming reduces each word to its root, which enables the grouping of different words that refer to similar concepts. For instance, plurals are turned into singulars (*doors* to *door*), and verb tenses are unified (*changes* to *chang*).
4) Finally, domain terms extraction and stopwords removal techniques are applied. To do this, software engineers provide two separate lists of terms: one list of

both single-word and multiple-word terms that belong to the domain and must be kept for analysis, and a list of irrelevant words that have no analysis value. Both kinds of terms can be automatically filtered in or out of the query.

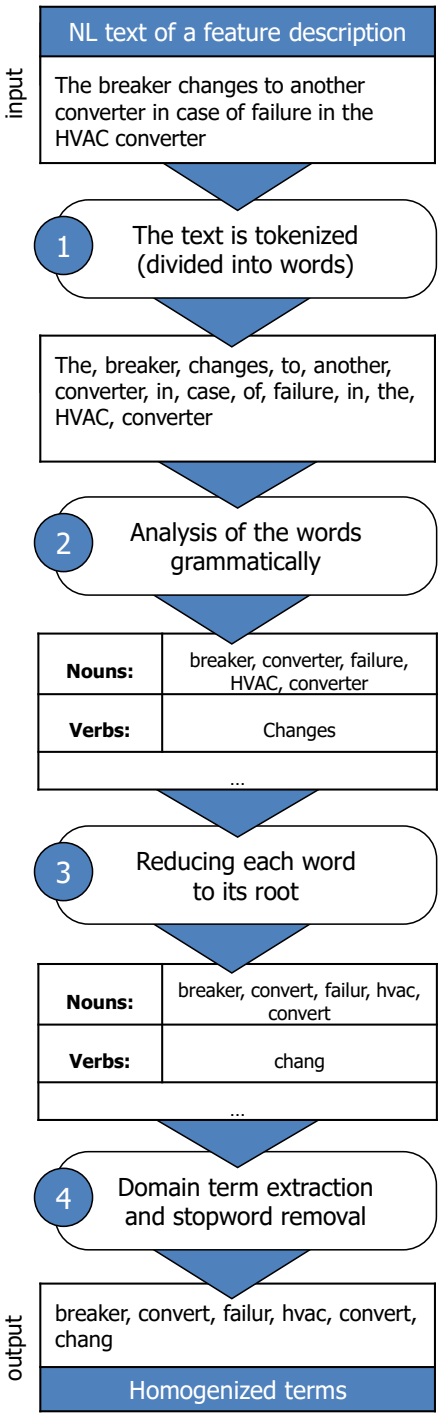


FIGURE 2: Example of text homogenization process for a feature description

Figure 2 shows an example of the text homogenization process using the NLP techniques described above (division into words, grammatical analysis of the words, root reduction, and finally, domain terms extraction and stopwords removal). The upper part of the figure shows the input of the text homogenization process, which is the NL text that describes the feature with the name "converter assistance". The feature description is as follows: *'The breaker changes to another converter in case of failure in the HVAC converter'*. The bottom part of the figure shows the homogenized terms that are obtained as output: *breaker, convert, failur, hvac, convert, chang*.

B. QUERY EXPANSION

In the available literature [19], it is possible to find several query expansion techniques that propose the inclusion of terms that fit with the vocabulary of the software artifacts in use. Not all of these techniques can be applied in our work (for instance, model based corpus). Consequently, we selected three existing techniques. To that extent, we applied the following criteria: (1) we did not take into consideration those techniques that rely on sources that are external to the corpus, (such as the web or ontologies); (2) we discarded the techniques that depend on the relationships between the words that are present in NL, due to the fact that these relationships are not shared between the words that appear in software [20]; and (3) since our goal is to support the daily FL tasks of engineers, we disregarded non-practical techniques based on algorithms with elevated levels of computational complexity.

It is important to highlight that the three selected expansion techniques consider the top K documents from the list of documents that are relevant for the query. Afterwards, these techniques order the terms of these K documents in different ways, and then select the top N terms to expand the query.

The first of the techniques is based on Rocchio's expansion technique [21], which is possibly the most utilized query reformulation method [22]. It works by ordering the terms that compose the top K relevant documents through the study of the importance, relative to the corpus, of each term. To that extent, the following equation is used:

$$Rocchio = \sum_{d \in R} TfIdf(t, d) \tag{1}$$

where R is the collection of the top K relevant documents, d is one document in R, and t is one term in d. The first component (Tf) is the term frequency, i.e., the number of times t appears in d. It is an indicator of the importance of t in d, compared to the rest of the terms in the document. The second component (Idf) is the inverse document frequency, that is, the inverse of the number of documents in the corpus that contain t. It indicates the specificity of t for d.

The second technique that orders the terms in the top K documents is the Robertson Selection Value (RSV) [19], [23]. The RSV uses $TfIdf$ as part of its equation, as Rocchio's method does. However, the RSV also takes into

account the chance of a term occurring in a relevant document to determine its importance. The RSV is calculated as follows:

$$RSV = \sum_{d \in R} TfIdf(t,d)[p(t|R) - p(t|C)] \quad (2)$$

where C denotes the whole set of the documents in the corpus, R represents the collection of the top K relevant documents, d is one document in R, and t is one term in d. In the equation, $p(t|R)$ is the number of times that t appears in R, and $p(t|C)$ is the number of times that t appears in C.

Neither the first nor the second technique relies on the similarities among the terms and the query. The intent of both techniques is to use the first K documents that are relevant to the query in order to expand it with the most representative terms. In contrast, the third technique orders the terms in the top K relevant documents through the *Dice* similarity coefficient [19], which relates the terms of the documents with each term of the query by using as a basis their appearances within the same documents in the corpus. The *Dice* similarity coefficient is given by the following equation:

$$Dice = \frac{2df_{u \wedge v}}{df_u + df_v} \quad (3)$$

where u is a term from the query, v is a term from the top K documents, and df denotes the number of documents in the corpus that contain either u (df_u), v (df_v), or both u and v ($df_{u \wedge v}$).

In our approach, we use the feature descriptions that are provided by the internal crowd as the relevant documents, which are used for query expansion. To select the top K relevant documents, our approach orders these feature descriptions from highest to lowest level of confidence.

An example of query expansion is shown in Figure 3. The upper part of the figure shows three different descriptions for the same target feature. One feature description is provided by the Requester (i.e., query), whereas two feature descriptions are provided by workers, who contribute to the internal crowdsourcing (i.e., relevant documents). The middle part of the figure shows both the result of the text homogenization process (as described in Subsection III-A), and the result of ordering the homogenized terms of the relevant documents (from highest to lowest relevance) using Rocchio's method as automatic query reformulation technique. The lower part of Figure 3 shows the reformulated query, which extends the terms provided by the requester (*breaker, convert, failur, hvac, convert, chang*) with the top 5 homogenized terms provided by the workers. As a result, the reformulated query contains the following terms: *breaker, convert, failur, hvac, convert, chang, energi, provid, current, coverag, overload*.

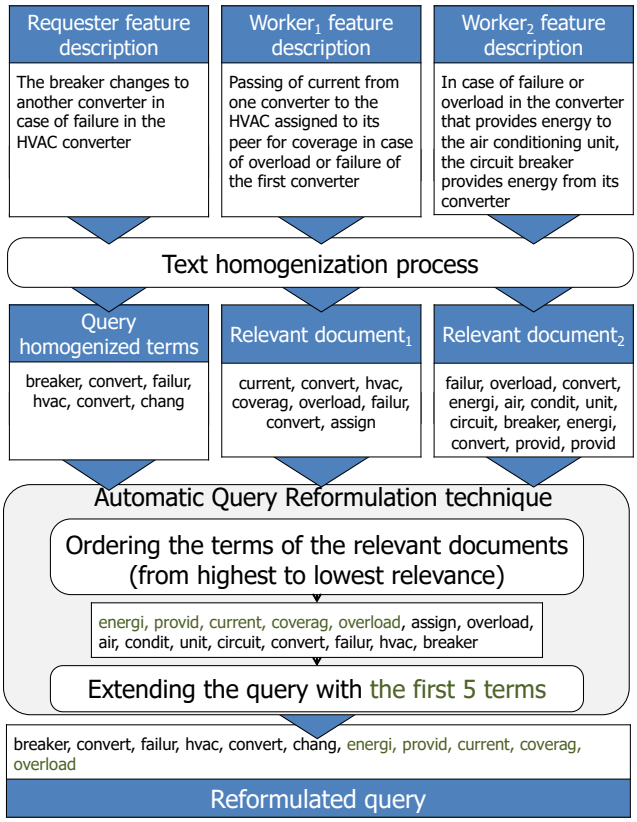


FIGURE 3: Example of query expansion

C. QUERY REDUCTION

Although queries get longer as more sophisticated information needs to be expressed, they tend to include noise among the relevant information. Thus, some terms in the query are only confusing the search engine. Hence, most commercial and academic search engines are affected by this issue, and their performance deteriorates when longer queries must be handled [24]. The aim of query reduction techniques, therefore, is to tackle this issue by shortening queries.

The query reduction technique adopted in this work is a conservative automatic reduction technique, previously applied in other software engineering works [17], [25]. The technique eliminates non-discriminating terms, i.e., terms appearing in more than 25% of the total of the documents that conform the corpus.

IV. FEATURE LOCATION TECHNIQUE

In order to perform FL in an automated fashion, Latent Semantic Indexing (LSI) [10] is used in previous works that locate features in code [4], [5] and it is the technique that obtains the best results for FL tasks [11]–[13]. For this reason, we use LSI in our approach to locate the most relevant model fragment for a reformulated query that is provided as input.

LSI is an mathematical approach that automatically analyzes relationships between *queries* and textual *documents*. Firstly, the technique encodes a *query* and a set of textual *documents* through a *term-by-document co-occurrence matrix*, representing them as vectors afterwards. Then, the technique analyzes the relationships between vectors to produce a sim-

ilarity ranking between the *query* and each of the *documents*.

In our approach, the inputs for LSI are the reformulated query and a product model as search space. Figure 4 shows a sample product model taken from a real-world train of our industrial partner, which is expressed using a Domain Specific Language (DSL). The DSL that is used by our industrial partner has enough expressiveness to define the interactions between the pieces of equipment that compose a train. For understandability and legibility purposes, and also due to intellectual property rights concerns, Figure 4 graphically depicts a simplified, equipment-focused subset of the DSL. The product model of the figure comprises a set of model elements that represent a scenario where two separate pantographs (high voltage equipment) collect energy to send it through voltage converters, which in turn feed the consumer equipment (for instance, the HVAC, which is the air conditioning system of the train).

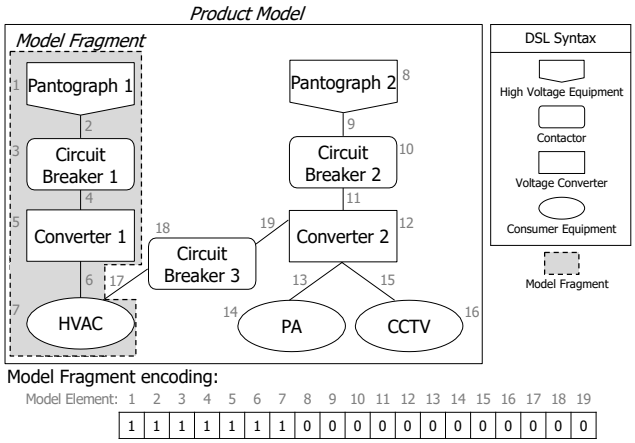


FIGURE 4: Product model and model fragment example

Figure 4 also shows in gray an example model fragment, which realizes a feature. A model fragment is composed of a subset of model elements that are part of the product model (including relationships). To manipulate the model fragment (which always belongs to the product model that is provided as search space), it is encoded using a string of binary values that contains as many positions as elements in the product model. Each position in the string has two possible values: 0 in case the model element does not appear in the model fragment, or 1 in case the model element does appear in the model fragment. In Figure 4, elements 1-7 comprise the model fragment, so the corresponding values are set to '1' in its binary string representation as the lower part of the figure shows.

Since the results of FL through LSI depend greatly in the style in which the NL of software artifacts is written [17], it is often regarded as beneficial to preprocess the inputs of LSI through NL processing techniques. Therefore, our approach preprocesses the NL of each model element of the input product model as described in Subsection III-A. For example, *HVAC* is a homogenized term that is extracted from *Model Element 7* of Figure 4.

The upper part of Figure 5 shows an example of *term-by-document co-occurrence matrix*. The rows in the matrix (*keywords*) represent the words (i.e., homogenized terms) that compose the inputs. The columns (*documents*) represent the model elements of the input product model. The last column represents the reformulated *query* introduced as input. Cells in the matrix contain the appearance frequency of the tex-titkeyword denoted by the row in the *document* denoted by the column.

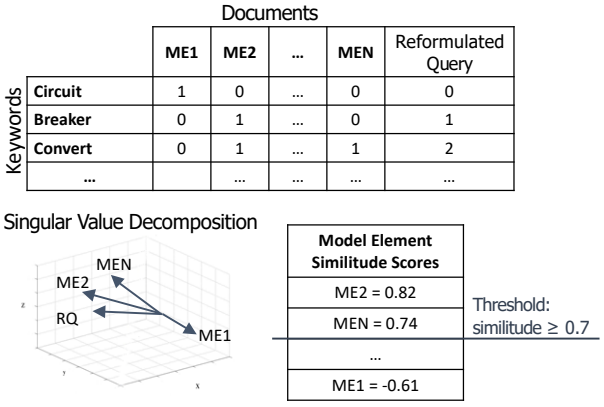


FIGURE 5: Automated Feature Location through Latent Semantic Indexing

Vector representations of the *documents* and the *reformulated query* are obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using *Singular Value Decomposition* (SVD) [10]. The lower-left part of Figure 5 depicts a three-dimensional graph of the SVD, containing the vectorial representations of some of the matrix columns and the *reformulated query*. SVD usually calculates the similarity between vectors using the cosine [10]. For this reason, the similarity degree between vectors is calculated as follows:

$$similarity(me) = \cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

where the similarity of a model element (me) is calculated as the cosine of the angle θ; the angle formed between the vector representing the latent semantic of the model element (A) and the vector representing the latent semantic of the reformulated query (B). 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*).

After performing the calculation, only those model elements with an associated similarity value greater than x must be taken into account. A widely used threshold for x is $x = 0.7$, corresponding to a $45°$ angle between vectors. The selection of this threshold is an issue that is still under study. However, the chosen value has led other works in the field to [26], [27] promising results. Therefore, in our approach, the model elements from the product model that was used

as search space that meet the $x \geq 0.7$ condition are set to '1' in a model fragment encoding and they conform a model fragment. This model fragment is candidate for realizing the reformulated query. In the example of Figure 5, *ME2* and *MEN* conform the model fragment. The model fragment generated in this manner is the final output of LSI.

V. EVALUATION

A. RESEARCH QUESTIONS

We aim to answer the following research questions to evaluate several aspects with regard to how the results are affected by enabling low-cost in internal crowdsourcing and using different automatic query reformulation techniques.

RQ_1: *Is there improvement in performance for internal crowdsourcing despite the limitation of time?*

RQ_2: *What is the performance locating features using different automatic query reformulation techniques in the context of internal crowdsourcing?*

RQ_3: *How much is the performance influenced using each automatic query reformulation technique?*

The first research question investigates in an industrial environment whether there are significant differences in the performance results (in terms of recall, precision and F-measure) of our approach, which limits the time in internal crowdsourcing, with a baseline approach, which does not use internal crowdsourcing. The second research question investigates the performance using different automatic query reformulation techniques. Finally, the third research question investigates how much the performance is influenced by comparing the results of each query reformulation technique with the baseline.

B. DATA SET

The data set in use is provided by CAF, our industrial partner. CAF is an international reference in the manufacturing of railway solutions, which can be found at a global level in many different shapes (trains, subway, light rail, or monorail, among others). The case study comprises 23 trains, with each model being composed by around 1200 elements, on average. The models are built from a basis of 121 different features, which can be part of a specific model. It is important to highlight that these features have been developed and maintained by different people over years.

For the evaluation of our approach, CAF provided us with two separate lists, one containing domain terms and one containing stopwords for NLP purposes. Moreover, CAF provided us with the names of 43 features from different trains, and a mapping to the model fragments that realize each feature. The model fragments comprise between 5 and 20 model elements. Nineteen CAF domain experts acted as either requester or worker for providing a description that is used to locate each of the 43 features.

C. IMPLEMENTATION DETAILS

We implemented the four automatic query reformulation techniques using Java. To manipulate the product models and to manage the model fragments, we used Eclipse Modeling Framework and CVL [28]. The NLP techniques were developed using the OpenNLP POS-Tagger [29] and the English (Porter2) stemming algorithm [30]. The Efficient Java Matrix Library (EJML [31]) was used to implement the LSI.

The evaluation was executed using a Dell XPS with a processor Intel(R) Core(TM) i7-2670QM @2.2GHz, 8 GB of RAM and Windows 10 Pro N 64 bits as the hosting Operative System. The time taken by the approach to locate each feature ranged between 10 seconds and 22 seconds.

D. PLANNING AND EXECUTION

Figure 6 presents an overview of the execution process and performance measurement that was planned to answer each research question. This process takes as input the documentation from our industrial partner (43 feature names, product models and model fragments of the features), 43 feature descriptions from 19 requesters, and the information provided by the internal crowd for each feature (18 workers who provide feature descriptions and their self-rated confidence within the time limit). Next, a baseline approach is executed in order to put the performance of our work in perspective. Moreover, four variants of our approach are executed where each variant uses a different automatic query reformulation technique.

As the result of the execution of either the baseline or one of the variants of our approach, we obtain a model fragment that realizes the target feature. The obtained model fragment is compared with the oracle as shown in Figure 6. The oracle is considered to be the ground truth (correct solutions for the issue under study), and is made up of the mapping between model fragments and target features provided by our industrial partner. By comparing the obtained results against the oracle, a confusion matrix is calculated. A confusion matrix is a table that describes the performance of a classification model on a set of test data (the best solutions) for which the true values are known (from the oracle). In our work, each obtained solution is a model fragment, composed by an assembly of model elements (subset of the complete model). Being the granularity at the model elements level, the presence or absence of each model element is considered as a true or false classification, respectively. The confusion matrix distinguishes between predicted values and real values by classifying them into four distinct categories:

1) True Positive (TP): predicted true in the solution, also true in the oracle.
2) False Positive (FP): predicted true in the solution, but false in the oracle.
3) True Negative (TN): predicted false in the solution, also false in the oracle.
4) False Negative (FN): predicted false in the solution, but true in the oracle.

From the confusion matrix, some performance measurements can be derived. Specifically, we report three performance measurements: recall, precision and F-measure.

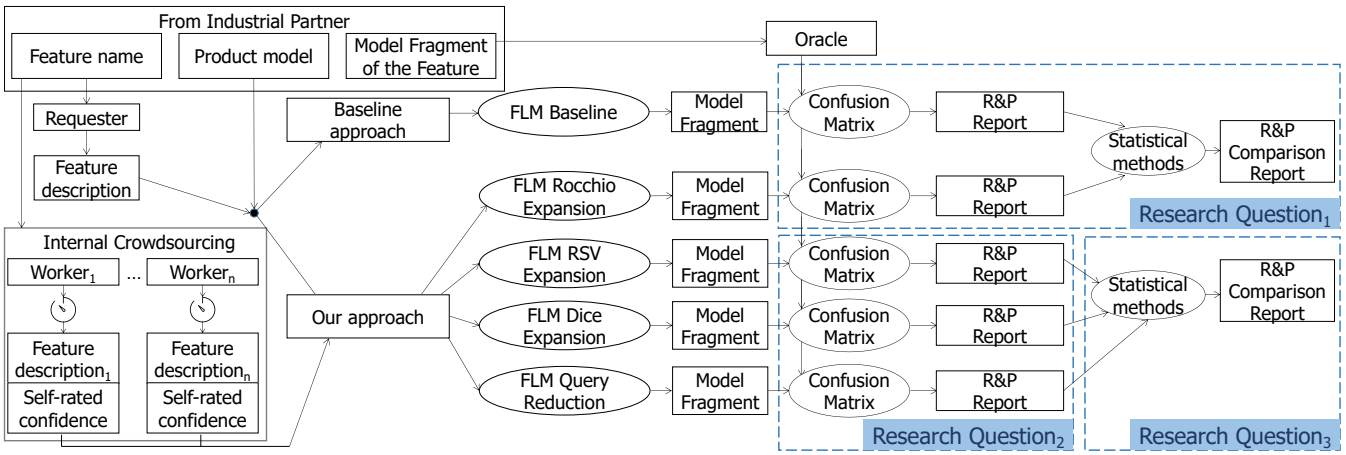


FIGURE 6: Evaluation execution and measurement to answer each research question

Recall measures the percentage of elements of the oracle correctly retrieved by the proposed solution, and is calculated as:

$$Recall = \frac{TP}{TP + FN}$$

Precision measures the percentage of elements from the proposed solution that appear in the oracle, and is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

The F-measure is the harmonic mean of precision and recall, and is calculated as:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Recall and precision values can range from 0% to 100%. Obtaining values of 100% precision and 100% recall would imply that the obtained solution and the oracle are equal. A recall value of 0% would indicate that the obtained solution does not contain a single model element of the oracle, and a value of 100% recall would mean that the obtained solution has correctly retrieved all the model elements of the oracle. A precision value of 0% would denote that no single model element from the obtained solution appears in the oracle, and a value of 100% precision would entail that all the model elements from the obtained solution appear in the oracle.

To obtain the performance measurements that answer each research question, the execution process is as follows:

Answering RQ$_1$. To answer whether there is an improvement in performance for internal crowdsourcing despite the limitation of time, we executed a baseline approach that does not support internal crowdsourcing to locate features in models. The baseline follows the approach as described in Figure 1 without using the inputs of the "Internal Crowdsourcing" part. Specifically, the baseline takes as input a feature description from a requester and it locates the model fragment that realize the feature description using LSI as described in Section IV.

To locate the model fragment that realizes each of the 43 features using low-cost in internal crowdsourcing, we executed the variant of our approach that uses the query expansion technique based on Rocchio since it is the most commonly used method for query reformulation. To perform the reformulation, we set k=5 and N=10 (i.e., the top five workers' feature descriptions are set as relevant documents and the top 10 term suggestions from these feature descriptions are used to expand the requester's feature description). These settings are based on recommendations found in the literature [19]. For each feature, we executed our approach for each of the 19 domain experts (one is a requester and the remaining 18 are workers who constitute the internal crowd. The sum for the baseline and the approach presented gives a total of 1634 independent runs, i.e., 43 (features) x 19 (requesters) = 817 independent runs x 2 (baseline + approach). For each of the 43 features in the baseline and in our approach, we record the mean values and standard deviations for recall, precision and F-measure.

In addition, we compare the results of the baseline and our approach through the usage of statistical methods that can prove the statistical significance of our work, i.e., that provide formal and quantitative evidence of the improvements posed by our research. To that extent, we perform a Holm's post hoc analysis, which compares the results in a pair-wise manner. The result is a probability value, or $p - value$. The $p - value$ can range between 0 and 1, but in order to be accepted by the research community as statistically significant, a $p - value$ must be under 0.05.

However, statistically significant differences can be obtained even if they are so small as to be of no practical value. Therefore, in addition to addressing whether the results of an approach are statistically better than those obtained by another approach, it is important to assess the magnitude of the improvement. In other words, it is important to measure how much does the solution obtained by one approach improve the quality of the solution obtained by another approach. To analyze this issue, *Effect size* measures are needed. For a non-

parametric effect size measure, we use Cliff's delta [32], [33]. Cliff's delta is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap, with values ranging from -1 to 1. For instance, when comparing distributions X and Y: a value of 0 means no difference between the two distributions; a value of -1 means that all samples in distribution X are lower than all samples in distribution Y; and a value of 1 means the opposite (all samples in X are higher than all samples in Y). In addition, threshold values can be defined [34] for the interpretation of Cliff's delta effect size ($|d| < 0.147 \rightarrow$ 'negligible'; $|d| < 0.33 \rightarrow$ 'small'; $|d| < 0.474 \rightarrow$ 'medium', $|d| \geq 0.474 \rightarrow$ 'large').

Answering RQ$_2$. To measure the performance of locating features using different query reformulation techniques in the context of internal crowdsourcing, we executed three variants of our approach where each variant uses a different automatic query reformulation technique (either RSV expansion, Dice expansion and Query Reduction). For each variant, we executed our approach to locate each of the 43 features using the information provided by each of the 19 domain experts (one is a requester and the remaining 18 are workers who constitute the internal crowd), i.e., 3 (variants) x 43 (features) x 19 (requesters) = 2451 independent runs. For each variant and each of the 43 features, we record the mean values and standard deviations for precision, recall and F-measure.

Answering RQ$_3$. To answer how much is the performance influenced using each automatic query reformulation technique, we perform a pair-wise comparison between the results of each query reformulation technique and the baseline (that does not support internal crowdsourcing as described in RQ$_1$) using statistical methods. Thus, formal and quantitative evidence about the influence of internal crowdsourcing using each automatic query reformulation technique can be provided. Specifically, we carry out a Holm's post hoc analysis and a Cliff's delta as described to answer RQ$_1$.

VI. RESULTS AND DISCUSSION

A. RESEARCH QUESTION 1

Table 1 shows the mean values of recall, precision and F-measure for the 43 features of the industrial case study. The baseline obtains the best result in terms of recall, which obtains a value of 70.28%. Our approach (using Rocchio as the query expansion technique) outperforms the baseline in terms of precision and F-measure (50.90% and 42.71%, respectively).

Table 2 shows the $p-Values$ of Holm's post hoc analysis of the pair-wise comparison between the performance indicators (recall, precision and F-measure) of our approach and the baseline. The results show that all $p-Values$ have statistically significant differences in the performance indicators since they are smaller than the corresponding significance threshold value (0.05).

TABLE 1: Mean values and standard deviations for recall, precision and F-measure for the baseline and our approach using Rocchio expansion in the industrial case study

	Recall $\pm$ (σ)	Precision $\pm$ (σ)	F-measure $\pm$ (σ)
Baseline	70.28 $\pm$ 10.00	16.51 $\pm$ 7.12	26.02 $\pm$ 9.43
Our approach using Rocchio	41.21 $\pm$ 14.81	50.90 $\pm$ 13.45	42.71 $\pm$ 11.34

Table 2 also shows the Cliff's Delta values, which can be interpreted as large according to the magnitude scales [34]. Specifically, the Cliff's Delta value in precision shows large differences of our approach with regard to the baseline (0.9892). In addition, the Cliff's Delta value in the harmonic mean of precision and recall (F-measure) shows large differences of our approach with regard to the baseline (0.7555).

TABLE 2: Statistical analysis for comparing our approach using Rocchio expansion with the baseline

Our approach using Rocchio expansion vs Baseline					
Holm's post hoc			Effect size		
$p-Value$			Cliff's Delta		
Recall	Precision	F-measure	Recall	Precision	F-measure
$1.3x10^{-13}$	$2x10^{-14}$	$5.5x10^{-9}$	-0.9027	0.9892	0.7555

RQ$_1$ answer. From the results, we can conclude that low-cost in internal crowdsourcing when locating features in models produces an improvement in performance despite the limitation of time.

B. RESEARCH QUESTION 2

Table 3 shows the mean values of recall, precision and F-measure for the 43 features of the three variants of our approach. Each variant uses a different automatic query reformulation technique (either RSV expansion, Dice expansion or Query Reduction).

TABLE 3: Mean values and standard deviations for recall, precision and the F-measure for our approach using different automatic query reformulation techniques

	Recall $\pm$ (σ)	Precision $\pm$ (σ)	F-measure $\pm$ (σ)
RSV expansion	37.67 $\pm$ 14.65	52.12 $\pm$ 12.32	41.23 $\pm$ 11.59
Dice expansion	52.42 $\pm$ 14.19	59.10 $\pm$ 15.01	53.68 $\pm$ 11.32
Query Reduction	28.45 $\pm$ 14.55	81.33 $\pm$ 12.52	39.88 $\pm$ 16.59

RQ$_2$ answer. Reduction obtains the best result for precision, providing a precision value of 81.33%, whereas the worst precision value is obtained by RSV expansion (52.12%). Dice expansion obtains the best result for recall and F-measure (52.42% and 53.68%, respectively), whereas the worst result for recall and F-measure is obtained by Query Reduction (28.45% and 39.88%, respectively).

C. RESEARCH QUESTION 3

Table 4 shows the $p-Values$ of Holm's post hoc analysis for pair-wise comparisons between each reformulation technique (RSV expansion, Dice expansion and Query Reduction) and the baseline (as presented in the results of RQ_1 in Subsection VI-A) for recall, precision and F-measure. The results show that all pair-wise comparisons have statistically significant differences because the values are smaller than the corresponding significance threshold value (0.05). With regard to the Cliff's Delta values, Table 5 shows that the baseline obtains large differences in recall, whereas all automatic query reformulation techniques (either expansion or reduction) show large differences with the baseline in precision and F-measure according to the magnitude scales [34]. In F-measure, the highest Cliff's Delta value is obtained when Dice expansion is compared to the baseline (with the value of 0.9427).

TABLE 4: Holm's post hoc $p - Values$ to compare each reformulation technique with the baseline

	Recall	Precision	F-measure
RSV Expansion vs Baseline	$9.8x10^{-14}$	$1.5x10^{-14}$	$9.3x10^{-11}$
Dice Expansion vs Baseline	$2.4x10^{-7}$	$2x10^{-14}$	$5.9x10^{-14}$
Query Reduction vs Baseline	$1.5x10^{-14}$	$1.5x10^{-14}$	$1.2x10^{-05}$

TABLE 5: Cliff's Delta values to measure the effect size between each reformulation technique and the baseline

	Recall	Precision	F-measure
RSV Expansion vs Baseline	-0.9448	0.9957	0.7015
Dice Expansion vs Baseline	-0.6917	0.9957	0.9427
Query Reduction vs Baseline	-0.9935	1	0.5068

RQ_3 answer. From the results, we can conclude that the influence in the performance indicator of F-measure (the harmonic mean of precision and recall) that is obtained by our approach using query reformulation techniques (Rocchio expansion, RSV expansion, Dice expansion and Query reduction) is both significant and large.

D. DISCUSSION

In an industrial context where engineers' availability is scarce, our empirical results have confirmed that low-cost in internal crowdsourcing improves significantly the results of locating features in models. Although the results do not include all of the model elements of the features (100% of recall and precision), the different variants of our approach obtain large differences with the baseline in precision and F-measure.

For RQ_1 (using Rocchio as the query reformulation technique), the result of 50.90% in precision of our approach is considered excellent according to the classification obtained from [35]. We acknowledge that the classification obtained from [35] is to trace requirements in code. To the best of our knowledge, there is no other table that classifies the results of recall and precision for locating features in models.

It is important to highlight that the results for RQ_1 of our approach show statistically significant differences with regard to the baseline. For Precision and F-measure (the harmonic mean of precision and recall), our approach also shows large differences with regard to the baseline (according to the Cliff's Delta value). Despite the limitation of time, the results point that low-cost in internal crowdsourcing is promising. In addition, the results obtained with other reformulation techniques can lead to better results as it is shown in RQ_2 with Dice expansion.

Engineers can consider the model fragment provided by our approach as an initial solution. Since the feature location depends on the terms of the reformulated query, the initial solution can be manually refined (in case that model elements do not meet the engineers' expectations), or engineers may refine their terms of feature descriptions to automatically obtain different solutions.

Using our approach, engineers benefit from techniques that automatically support both low-cost internal crowdsourcing and feature location. The limitation of time that is proposed in our approach reduces the effort during software maintenance and evolution tasks and improve the quality of the solution. In the context of our industrial partner, suppose we ask the engineer to manually locate the model elements that correspond to the 43 features of the data set provided by CAF. Taking into account that the data set comprises 23 trains and the product model of each train has more than 1200 model elements, at least 27600 model elements should be evaluated. To assess a model element, it is reasonable to consider its properties. In the data set, each element has about 15 properties. Therefore, about 414000 properties of model elements should be considered. Assuming that an engineer only needs 1 second to consider a property of a model element, the engineer needs 4.79 days to manually locate each feature. In this industrial context where maintenance contracts last 25 years, an engineer may not have the full knowledge to locate the features in the product model. Therefore, the manual result of the location may not be correct and time-consuming. For this reason, techniques that automatically support both low-cost in internal crowdsourcing and feature location are necessary.

We detected that the support of low-cost in internal crowdsourcing of our approach improves the results because the reformulated query can include more appropriate terms to locate the target feature. These results are promising, and they promote that low-cost in internal crowdsourcing becomes a widespread practice in FL. As well as in other works [22], [36], results depend on the quality of the queries. By analyzing the results, we detected that the quality of the solution is negatively influenced due to the following issues: 1) engineers omitted terms in the queries due to a lack of knowledge, and 2) the language used in the queries was different to the language used in the models because the models were maintained by different engineers over years. Addressing these issues to evaluate the influence in the quality of the solution as well as evaluating new techniques

and new combinations of workers constitute our future work.

VII. THREATS TO VALIDITY

In order to acknowledge the limitations of the evaluation of our work, we use the classification of threats of validity presented in [37], [38], that distinguishes four aspects of validity:

Construct validity: our evaluation is performed through the usage of three measurements (recall, precision, and F-measure), widely accepted in the software engineering research community [39], to minimize this risk.

Internal Validity: to evaluate our approach and the baseline, we used a set of feature names and an oracle as ground truth (obtained from our industrial partner) that specified the solution that is considered as correct and that enables the calculations of recall, precision, and F-measure. Regarding the number of relevant documents and the number of terms in use to expand the query, we used the values of 5 and 10, respectively, as recommended in the literature [19]. With regard to the time limit that workers have to provide both the feature descriptions and the self-rated level of confidence, we set 10 minutes as decided by the workers of our industrial partner. However, at this stage of our research, we have not studied how different time values would impact the results yet.

External Validity: to mitigate this threat, we have designed our approach in such a way that allows for it to be applied not only to the domain of our industrial partner, but to other different domains as well. To apply our approach, the only requirements are: (1) that the set of models where features must be located conform to MOF (the OMG metalanguage for defining modeling languages); (2) that feature descriptions must be provided in a textual manner; and (3) that the self-rated confidence level must be provided as a number ranging from 1 to 7.

Furthermore, automatic query reformulation techniques only work if a feature description is used as a query, and in cases where the feature descriptions formulated by the internal crowd are well worded enough to retrieve at least some of the relevant documents [22]. As well as in other works [22], [36], results depend on the quality of the queries: poor queries cause irrelevant model fragments to be highly ranked. It is also worth noting that the language used for the textual elements of the models and for the feature descriptions must be the same. Eventually, some modifications can be applied to the NLP techniques to narrow the gap between both elements (different tokenizers, stemming, or POS tagging techniques). In this work, we select those NLP techniques that obtain the best results in a previous work [40]. Since the contribution of this work is to evaluate low-cost in internal crowdsourcing, we have not yet studied how different NLP techniques impact the results. Note that the language in use is particular for each domain, and therefore, even though our approach can be applied to locate features in MOF-based models from a real-world industrial context, the approach should be applied to other domains before completely reassuring its generalization.

Reliability: to reduce this threat, concerning to the extent with which the data and the analysis are dependent on the researchers, both the feature descriptions and the models are provided by our industrial partner, who is not further involved in the development of this research work.

VIII. RELATED WORK

Previous works [6]–[9] have been proposed to use external crowdsourced knowledge to reformulate queries for code search. Specifically, the external crowdsourced knowledge is obtained from StackOverflow Q&A site. However, these works are focused on searching in code instead of searching in other software artifacts such as models. Moreover, these works use external crowdsourced knowledge, which cannot be available to obtain relevant information in specific industrial contexts (e.g., due to intellectual property rights concerns).

Other works propose to expand the query in an automatic fashion to add words that are similar to or related to the query terms. Hill et al. [36] use word context to extract potential query expansion terms from the code. Other approaches propose to reformulate queries by removing words. Kumaran and Carvalho [24] reduce queries by analyzing the most promising subsets of terms from the original query. Haiduc et al. [17] propose an approach trained with a sample of queries and relevant results to automatically recommend a reformulation technique (expansion or reduction). Other works have been proposed to improve the effectiveness of FL by involving the feedback of users regarding the relevance of the retrieved results. For instance, Wang et al. [41] proposed a code search approach, incorporating user feedback to refine the query. Zou et al. [42] proposed a re-ranking approach that refines search results by investigating the style of the answers to software questions with different interrogatives. Despite the effort of improving the performance of FL in code through automatic query reformulation, it has been neglected in models and in industry.

In our previous works [43]–[45], we also perform query reformulation to locate features in models. In [43], we use the common approach for query reformulation by using as relevant documents other train models instead of using feature descriptions produced by software engineers as this work does. In this paper, the results are better than without the internal knowledge provided by software engineers. In [44], we obtain a reformulated query from a set of domain experts that serves to evaluate two techniques (Information Retrieval using LSI and Linguistic rules) for locating relevant model fragments. Since LSI obtains the best results in [44], we use LSI in this paper as the feature location technique. In [45], we propose CoFLiM and we evaluate whether collaboration improves the quality of the solution as well as how the quality of the solution is influenced by the number of domain experts input. CoFLiM only uses Rocchio expansion as the query reformulation technique and it uses a different FL technique

(evolutionary algorithm), whereas this paper evaluates the quality of the solution using Rocchio expansion as well as other three query reformulation techniques (RSV expansion, Dice expansion and Query Reduction) using a different FL technique (LSI). In contrast to [44], [45], this paper compares the results with a baseline and explores how the quality of the results is affected by: (1) limiting the time that the internal crowd has to provide knowledge (which is a need to promote internal crowdsourcing in industrial environments), and (2) using different automatic query reformulation techniques.

IX. CONCLUDING REMARKS

Despite the issues that engineers have for locating features and the crowdsourcing benefits, internal crowdsourcing is not a widespread practice. Engineers are busy resources and their time is a valuable asset in industry, which hinders engineers' willingness to invest in internal crowdsourcing. In this paper, we propose a novel approach that is a low-cost variant of internal crowdsourcing in software engineering for locating features in models. Our approach limits the time that engineers can spend for providing knowledge. After, our approach uses this knowledge to automatically reformulate an initial feature description. Finally, the reformulated feature description is taken as input to automatically locate the relevant model fragment using LSI as FL technique.

We evaluate four variants of our approach in a real-world case study to explore different automatic query reformulation techniques (Rocchio query expansion, RSV query expansion, Dice query expansion or Query reduction) to locate features in models. In addition, we evaluate a baseline approach, which does not use internal crowdsourcing, to compare the results of our approach using statistical methods. The results of our approach show that Dice query expansion obtains the best result in F-measure (53.68) instead of Rocchio query expansion (42.71), which is the most commonly used method for query reformulation. The results also show that the superiority of our approach is significant and large compared to the results of the baseline. Specifically, our approach improves F-measure up to 27.66%.

As future work, we plan to investigate how much the performance is influenced by changing (either increase or decrease) the time limit. Moreover, we plan to evaluate the influence in the quality of the solution with new approaches to locate the relevant model fragment (e.g., using Machine Learning), and to recruit a different combination of workers (e.g., increasing the number of workers) who can provide other knowledge to reformulate the initial feature description.

ACKNOWLEDGEMENTS

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 ALPS (RTI2018-096411-B-I00).

REFERENCES

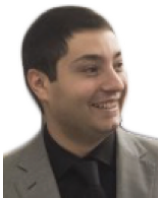
[1] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," Journal of Systems and Software, vol. 126, pp. 57 – 84, 2017.
[2] L. Favre, "Modernizing software & system engineering processes," in Proceedings of the International Conference on Systems Engineering, 2008, pp. 442–447.
[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey." Journal of Software: Evolution and Process, vol. 25, no. 1, pp. 53–95, 2013.
[4] J. Rubin and M. Chechik, "A survey of feature location techniques," in Domain Engineering. Springer, 2013, pp. 29–58.
[5] A. Razzaq, A. Le Gear, C. Exton, and J. Buckley, "An empirical assessment of baseline feature location techniques," Empirical Software Engineering, vol. 25, no. 1, pp. 266–321, 2020.
[6] M. M. Rahman and C. K. Roy, "Quickar: Automatic query reformulation for concept location using crowdsourced knowledge," in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 220–225.
[7] M. M. Rahman and C. Roy, "Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics," in Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), 2018.
[8] Z. Li, T. Wang, Y. Zhang, Y. Zhan, and G. Yin, "Query reformulation by leveraging crowd wisdom for scenario-based software search," in Proceedings of Internetware, 2016, pp. 36–44.
[9] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," IEEE Transactions on Services Computing, vol. 9, no. 5, pp. 771–783, 2016.
[10] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," Discourse processes, vol. 25, no. 2-3, pp. 259–284, 1998.
[11] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 420–432, Jun. 2007.
[12] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "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, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 234–243.
[13] M. Revelle, B. Dit, and D. Poshyvanyk, "Using data fusion and web mining to support feature location in software," in IEEE 18th International Conference on Program Comprehension (ICPC), 2010, pp. 14–23.
[14] K. Finstad, "Response interpolation and scale sensitivity: Evidence against 5-point scales," Journal of usability studies, vol. 5, no. 3, pp. 104–110, 2010.
[15] X. A. Lu and R. B. Keefer, "Query expansion/reduction and its impact on retrieval effectiveness," in Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994, 1994, pp. 231–240.
[16] S. Winkler and J. Pilgrim, "A Survey of Traceability in Requirements Engineering and Model-Driven Development," Software and Systems Modeling (SoSyM), vol. 9, no. 4, pp. 529–565, 2010.
[17] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13, 2013, pp. 842–851.
[18] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in Proceedings of the 2003 conference on Empirical methods in natural language processing, 2003, pp. 216–223.
[19] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," ACM Comput. Surv., vol. 44, no. 1, pp. 1:1–1:50, Jan. 2012.
[20] G. Sridhara, E. Hill, L. L. Pollock, and K. Vijay-Shanker, "Identifying word relations in software: A comparative study of semantic similarity tools." in ICPC, R. L. Krikhaar, R. LÃd'mmel, and C. Verhoef, Eds. IEEE Computer Society, 2008, pp. 123–132.
[21] G. Salton, The SMART Retrieval System—Experiments in Automatic Document Processing. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1971.
[22] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in Proceedings of the 10th Working

Conference on Mining Software Repositories, MSR '13, 2013, pp. 309–318.
[23] S. E. Robertson, "On term selection for query expansion," Journal of Documentation, vol. 46, no. 4, pp. 359–364, 1990.
[24] G. Kumaran and V. R. Carvalho, "Reducing long queries using query quality predictors," in Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '09. New York, NY, USA: ACM, 2009, pp. 564–571.
[25] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in ir-based concept location." in ICSM. IEEE Computer Society, 2009, pp. 351–360.
[26] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in Proceedings of the 11th Working Conference on Reverse Engineering, ser. WCRE '04, 2004, pp. 214–223.
[27] H. E. Salman, A. Seriai, and C. Dony, "Feature location in a collection of product variants: Combining information retrieval and hierarchical clustering," in The 26th International Conference on Software Engineering and Knowledge Engineering, 2013, pp. 426–430.
[28] Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding standardized variability to domain specific languages," in Software Product Line Conference, 2008. SPLC '08. 12th International, Sept 2008, pp. 139–148.
[29] "Apache opennlp: Toolkit for the processing of natural language text," https://opennlp.apache.org/, 2017.
[30] "English (porter2) stemming algorithm," http://snowball.tartarus.org/algorithms/english/stemmer.html, 2018.
[31] "Efficient java matrix library," http://ejml.org/, 2017.
[32] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." Psychological Bulletin, vol. 114, no. 3, p. 494, 1993.
[33] ——, Ordinal methods for behavioral data analysis. Psychology Press, 1996.
[34] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohenâĂŹsd for evaluating group differences on the nsse and other surveys," in annual meeting of the Florida Association of Institutional Research, 2006, pp. 1–33.
[35] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," IEEE Transactions on Software Engineering, vol. 32, no. 1, pp. 4–19, Jan 2006.
[36] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of nl-queries for software maintenance and reuse," in Proceedings of the 31st International Conference on Software Engineering, ser. ICSE '09, 2009, pp. 232–242.
[37] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical software engineering, vol. 14, no. 2, pp. 131–164, 2009.
[38] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in software engineering, 2012.
[39] G. Salton and M. J. McGill, "Introduction to Modern Information Retrieval," 1986.
[40] R. Lapeña, J. Font, O. Pastor, and C. Cetina, "Analyzing the impact of natural language processing over feature location in models," in Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2017, 2017, pp. 63–76.
[41] S. Wang, D. Lo, and L. Jiang, "Active code search: Incorporating user feedback to improve code search relevance," in Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ser. ASE '14, 2014, pp. 677–682.
[42] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, and L. Zhang, "Learning to rank for question-oriented software text retrieval," in Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, 2015, pp. 1–11.
[43] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Automatic query reformulations for feature location in a model-based family of software products," Data & Knowledge Engineering, 2018.
[44] F. Pérez, A. C. Marcén, R. Lapeña, and C. Cetina, "Introducing collaboration for locating features in models: Approach and industrial evaluation," in Proceedings of the 25th International Conference on Cooperative Information Systems, CoopIS, 2017, pp. 114–131.
[45] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," Automated Software Engineering, 2019.

Francisca Pérez is an assistant professor (tenure track) in the SVIT Research Group (https://svit.usj.es) at San Jorge University. She received a PhD in Computer Science from the Technical University of Valencia. Her research interests include Model-Driven Development, Collaborative Information Retrieval, Search-Based Software Engineering, and Variability Modeling. She publishes her research results and participates in high quality international software engineering conferences and journals, such as the Automated Software Engineering (AUSE) journal, the Information & Software Technology (IST) journal, and the Journal of Systems and Software (JSS). More about Pérez and her work is available online at http://franciscaperez.com.

Ana C. Marcén is a PhD student in computer science and researcher in the SVIT Research Group at San Jorge University. Her current research lines include feature location, trazability link recovery and machine learning. Marcén obtained her Masters' degree in computer science from San Jorge University. Contact her at acmarcen@usj.es

Raúl Lapeña is a PhD student in computer science and researcher in the SVIT Research Group at San Jorge University. His main research interests lie in model-driven development, feature location, and software product lines. Lapeña received his degree in computer engineering from San Jorge University. Contact him at rlapena@usj.es



Carlos Cetina is an associate professor with San Jorge University and the Head of the SVIT Research Group. He received a PhD in computer science from the Technical University of Valencia. His research focuses on software product lines and model-driven development. His research results have reshaped software development in world-leader industries from heterogeneous domains ranging from induction hob firmware to train control and management systems. More information about his background can be found at his website: http://carloscetina.com.