# How the Quality of Maintenance Tasks is Affected by Criteria for Selecting Engineers for Collaboration

Francisca Pérez, Raúl Lapeña, Ana C. Marcén, Carlos Cetina

**Abstract**—In industrial software projects, projects might span over decades and many engineers join or leave the company over time. For these reasons, no single engineer has all of the knowledge when maintenance tasks such as Traceability Link Recovery (TLR), Bug Localization (BL), and Feature Location (FL) are performed. Thus, collaboration has the potential to boost the quality of maintenance tasks since the solution of an engineer might be enhanced with contributions of solutions from other engineers. However, assembling a team of software engineers to collaborate may not be as intuitive as we might think. In the context of a worldwide industrial supplier of railway solutions, this work evaluates how the quality of TLR, BL, and FL is affected by the criteria for selecting engineers for collaboration. The industrial supplier uses software models to generate the software that controls their trains. The criteria for collaboration are based on engineers' profile information to select the set of search queries that are involved in the maintenance task. Collaboration is achieved by applying automatic query reformulation, and the location of the relevant model fragment for the requirement/bug/feature being located relies on an Evolutionary Algorithm. We assess the quality of the retrieved model fragment in terms of recall, precision, and F-measure. We provide evidence of the significance of the results by means of statistical analysis. A focus group confirmed the relevance of the findings. Our work uncovers how software engineers who might be seen as not being relevant in the collaboration (because of low confidence levels) can lead to significantly better results.

**Index Terms**—Collaborative Software Engineering, Search-Based Software Engineering, Model-Driven Engineering

## 1 Introduction

Software maintenance is a challenging activity in industrial environments where a vast number of software artifacts are accumulated over the years and these artifacts have been created and maintained by different software engineers. Since no single software engineer has a full understanding of the entirety of the software artifacts, several software engineers can collaborate to complement the knowledge that each one has of the artifacts [1].

Previous works [2], [3], [4], [5], [6] address collaboration using external knowledge to reformulate an individual's queries for code location. Specifically, this external knowledge is obtained from the Stack Overflow Q&A site. However, external knowledge may not be available to obtain relevant information in specific industrial contexts (e.g., due to intellectual property rights concerns). Therefore, in these contexts, collaboration should be performed among the software engineers who work on maintaining the software products. However, software engineers are confronted with the following question once they decide to collaborate: What profile should the software engineers involved in the collaboration have?

In this paper, we evaluate how the criteria for the selection of software engineers influence the quality of maintenance solutions that are obtained as a result of collaboration. In our work, we address the following maintenance tasks: Traceability Link Recovery (TLR), Bug Localization (BL), and Feature Location (FL). We address these maintenance tasks because they are among the most common and relevant tasks in the Software Engineering field, especially when maintaining software products [7], [8], [9], [10].

We have built on industrial experience through the participation of software engineers to address the maintenance tasks. The software engineers are from three different distributed teams of an industrial partner. The industrial partner, *Construcciones y Auxiliar de Ferrocarriles* (CAF)[1], is a worldwide supplier of railway solutions that has developed a family of PLC software to control the trains they have manufactured for more than 25 years. To develop this software, the industrial partner uses software models for code generation following the ideas of Model Driven Software Development [11]. We acknowledge that software models have not replaced source code as a means of software development, but they have nonetheless been reported as a successful paradigm for developing industrial software [12].

In our evaluation, each engineer (19) produces a search query and a profile for the following: each requirement for TLR (50), each bug description for BL (42), and each feature name for FL (43). For a total of 2565 search queries and profiles. The profile is in terms of model ownership, self-rated confidence level, and number of days since the last modification in the model. The collaboration takes into

• *F. Pérez, R. Lapeña, A. C. Marcén and C. Cetina are with the SVIT Research Group of Universidad San Jorge, Zaragoza, Spain. E-mail: {mfperez, rlapena, acmarcen, ccetina}@usj.es. C.Cetina is also with University College London, London, United Kingdom.*

*Manuscript received November X, 2021; revised .*

1. www.caf.net/en

account the pro les for the selection of the participants' queries. Once the queries are selected, collaboration is achieved by applying automatic query reformulation [13], [14], and the search for the relevant model fragment relies on an Evolutionary Algorithm that is guided by similitude to the resulting query [15], [16]. This Evolutionary Algorithm establishes the model fragment that implements a speci c requirement (TLR), identi es the model fragment that causes a particular error (BL), or identi es the model fragment that is associated with a speci c functionality or characteristic (FL).

To assess the quality of the result, we compare the resulting model fragment with an oracle, which is the ground truth. From the comparison, we obtain a report with the following performance measures that are widely accepted in the software engineering research community [17]: recall, precision, and F-measure. Moreover, we provide evidences of the signi cance of the results by means of statistical analysis.

After analyzing the results, we have learned that the combination of software engineers who provide the best quality of the solutions may not be intuitive. Our industrial experience offers a new interpretation of the role that underdogs (i.e., software engineers who might be seen as not being relevant in the collaboration because of low con dence levels) can play in the collaboration:

> In TLR, engineers who do not belong to the owner team of the requirement should be involved. When dealing with non-familiar requirements, an engineer produces a more detailed search query, which mitigates the issue of tacit knowledge that the engineers who belong to the owner team have.
> In FL, instead of only requesting the con dence level, both the con dence level and an estimated coverage of the feature should be requested. Even though engineers report low con dence levels when they do not have knowledge of the whole feature, they have deep knowledge of a small part of the feature. Coverage estimations help to prevent this feature knowledge from being overlooked.

In addition, our results con rm the Defect Principle [18]. In BL, engineers who performed the latest modi cations should be prioritized.

A focus group acknowledged that the lessons learned to improve the selection of engineers for collaboration are counterintuitive, but they do lead to better results. No previous work has reported the positive in uence of underdogs on collaboration. Thus, more software engineers and researchers (as happened with the engineers of the industrial partner of this work) might be missing the potential of underdogs that this work uncovers.

The remainder of the paper is structured as follows: Section 2 introduces the industrial partner domain and explains the motivation for our work. Section 3 presents an overview of our work. Section 4 presents the real-world criteria for performing the selection of participants. Section 5 describes collaborative fragment retrieval on models. Section 6 describes the evaluation, and Section 7 presents the results. Section 8 presents the discussion and lessons learned. Section 9 describes the threats to validity. Section 10 presents related work. Finally, Section 11 concludes the paper.

## 2 Background and Motivation

This section introduces the Domain-Speci c Language (DSL) that our industrial partner uses to specify and generate the implementation code of their products. We also present the motivation for the need of our work. Figure 1 depicts a basic example of a model using an equipment-focused, simpli ed subset of the DSL of our industrial partner. It shows two separate pantographs (High Voltage Equipment) that 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 shown in the upper-right part of the gure (the train's air conditioning system), and the PA (public address system) and CCTV (television system) on the right.

Fig. 1. Example of a product model and a model fragment

Figure 1 also depicts (in gray) a set of model elements that belong to the product model. These model elements show an example of a model fragment, which is the realization of the feature: HVAC Assistance. This model fragment allows the passing of current from one converter to the HVAC that is assigned to its peer for coverage in case of overload or failure of the rst converter.

At this point, it is important to highlight that a model fragment is not extracted from its parent model as a new isolated model. The model fragment is used to identify elements of the model that are relevant for a requirement/bug/feature. This could be understood as highlighting/tagging model elements of the model (that is, no new artifact is created). Guided by the feature to be located, different combinations of model elements can be highlighted/tagged.

In addition, it is important to highlight the differences between a feature and a requirement. They are written in a different style, in a different phase of development, and with a different goal in mind. Requirements are written before development, are client-in uenced and are for contracts. In contrast, features are written when products already exist, are internal, and are for reuse.

Although the product model and the model fragment that realizes the feature of the example of Figure 1 makes feature location in models appear easy, it can become very

complex and time-consuming in models of industrial size. The DSL of our industrial partner addresses speci cation and code generation in a domain (software for railway control) where UML and SysML are also used for these particular tasks. For example, the data set provided by our industrial partner for feature location comprises 23 trains, and the model of each train has more than 1200 model elements. Therefore, 27600 model elements should be evaluated. In addition, it is reasonable to consider the properties of each model element since they hold domain knowledge. In the data set, each element has about 15 properties. Therefore, about 414000 properties of model elements should be considered, which is not viable even when assuming that an engineer only needs one second to evaluate a property.

Previous works [19], [20], [15] suggest the use of Search-based Software Engineering [21] to alleviate the above effort to locate model fragments. In these works, an evolutionary algorithm searches the space of model fragments to locate the relevant model fragments. The similarity between the text of each model fragment and a textual description (that is produced by a software engineer) guides the evolutionary algorithm. However, in industrial settings where products have been developed for 25 years (as is the case for the industrial partner), no single software engineer has knowledge of the entirety of the software models. Collaboration approaches extend the query produced by a software engineer with the queries of other software engineers. This collaboration by query expansions leads to an improvement in the quality of the results [16]. Nevertheless, the question of who should participate in the collaboration is still open, and this work contributes to addressing it.

## 3 Overview of our work

The aim of Fragment Retrieval on Models [15] is to obtain the most relevant model fragment (i.e., set of model elements) for a speci c TLR, BL, or FL query. To leverage collaboration, the query is obtained by automatically reformulating different software engineers' search queries [16]. In other words, the idea of collaboration in this paper is that of leveraging locally crowd-sourced information through mutual collaboration between engineers performing the software tasks.

This work evaluates the in uence of the selection criteria on the quality of the retrieved model fragment. The results will serve to recommend the pro le that software engineers should have in order to be involved in the collaboration of TLR, BL, and FL tasks.

Figure 2 presents an overview of our work. The left part of the gure shows the inputs from the industrial partner: 1) requirements for TLR, bug descriptions for BL, and feature names for FL; 2) the software models that are going to be used as search space; and 3) information input by software engineers.

For each requirement for TLR, bug description for BL, and feature name for FL provided by the industrial partner, each engineer provides a search query as input information. For each search query, the engineer also provides information about his/her pro le in terms of a self-rated con dence level (Likert scale). We request a self-rated con dence level to identify relevant search queries. In addition, for each search query, as pro le information the engineer provides the number of days from his/her last modi cation in the model fragment of the given requirement, feature name, or bug description. We request this information to identify relevant search queries since the Defect Principle (or Defect Localization Principle) states that the most recent modi cations to a project are the most relevant for certain Information Retrieval purposes [22], [18], [23].

Since the industrial partner has different teams of software engineers that perform maintenance tasks on models across different cities, each engineer also indicates whether his/her team owns the given requirement, feature, or bug. According to the industrial partner, the engineers consider that their team is the owner if the team has participated in the speci cation of the given requirement or feature in the software models.

Once the input information is provided, the engineers who participate in the collaboration are automatically selected (see the middle part of Figure 2). After the selection of participants for collaboration, their search queries are used as input to perform collaborative fragment retrieval on models as Figure 2, right shows. The result is the most relevant model fragment for the given type of query (natural language requirements for TLR, bug descriptions for BL, and feature descriptions for FL).

The next two sections describe the selection of participants for collaboration and collaborative fragment retrieval on models.

## 4 Selection of participants for collaboration

To perform the selection of participants for collaboration, we relied on real criteria that have been used in industry when maintenance tasks need to be performed. We selected these real criteria based on the experience of our industrial partner. Speci cally, we conducted interviews with software team leaders as well as brainstorming meetings with software engineers of our industrial partner to obtain the criteria. Below, we present each selected criterion from the most used to the least used by the industrial partner:

Criterion 1: Available owners. It selects software engineers who are available and belong to the team that is the owner of the requirement or feature, which is affected by the given requirement (for TLR), bug description (for BL) or feature name (for FL). Availability depends on many factors (e.g., work load, holidays, schedules, etc.), and it changes over time. Therefore, we selected random owners to emulate the real scenario of the industrial partner.

Criterion 2: The most con dent engineers. It selects software engineers who have the highest scores in self-rated con dence for the given requirement, bug description, or feature name. The software engineer provides the self-rated con dence level from 7 (highest self-rated con dence) to 1 (lowest self-rated con dence).

Criterion 3: The latest modi cations. It selects software engineers who have performed the most recent modi cations in the model fragment of the given requirement, bug description, or feature name. The time difference is based on the number of days and can therefore be very large when the model fragment was modi ed a long time ago. To normalize

Fig. 2. Overview of our work

the time difference, mathematical solutions such as square root or logarithm can be used. We used square roots because it has achieved good results in other works that use time differences [18].

Although the next criterion (gold criterion) has never been used by the industrial partner, the intuition of the industrial partner suggests that it would be the criterion that obtains the best results.

Gold criterion: The most con dent owners who performed the latest modi cation. This criterion selects software engineers who have the highest scores in self-rated con dence, perform the latest modi cations in the model fragment, and belong to the team that is the owner of the given requirement, bug description, or feature name.

The number of engineers to be selected is a con guration parameter (N) ranging from 2 to the maximum number of engineers who are considered for collaboration.

Other research works [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] have also reported that these criteria are being used in industry: available owners [26], [27], [28], con dence [29], [30], [31], [32], [33], and the latest modi cations [34], [35]. In other words, the industrial partner is not the only one that uses these criteria, but they are also relevant for other software developers. However, previous works have not yet compared these criteria as we have done.

## 5 Collaborative Fragment Retrieval on Models

This section describes how the collaborative fragment retrieval on models is done once the participants' search queries have been selected. To do this, Natural Language Processing techniques, automatic query reformulation, and fragment retrieval on models are used.

### 5.1 Natural Language Processing

The participants' search queries and all available text in the model elements are homogenized through Natural Language Processing (NLP) techniques since it is often regarded as bene cial and a frequent practice [36]. This homogenization is performed in the following steps using state-of-the-art NLP techniques. 1) The text is tokenized (i.e., divided into words). As a result, the text is divided using a white space since it is a tokenizer that can usually be applied. More complex tokenizers, such as CamelCase naming, need to be applied for some sources. 2) The Parts-of-Speech (POS) tagging technique analyzes the words grammatically and infers the role of each word in the text provided. As a result, each word is tagged in a category and some categories that do not provide relevant information (e.g., prepositions) can be removed. 3) Stemming techniques unify the language that is used in the text by reducing each word to its root. This serves to group together different words that refer to similar concepts. For instance, plurals are turned into singulars (circuits to circuit). 4) The Domain Term Extraction and Stopword Removal techniques are applied to automatically lter kind of terms in or out of the queries. To apply 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 that must be kept for analysis, and a list of irrelevant words that have no analysis value.

For example, the following feature description of the industrial partner "The breaker changes to another converter in case of failure in the HVAC converters" is homogenized into the following terms: breaker, convert, failur, hvac, convert, chang

### 5.2 Automatic Query Reformulation

Once the participants' search queries are homogenized, we apply automatic query reformulation to automatically combine the participants' search queries in a single query. Several query expansion techniques have been proposed to expand a query by adding terms [37], but not all of these techniques can be applied in our work because of the following: they do not support a model-based corpus; they rely on word relationships that exist in NL because in software words do not share the same relationships [38]; they rely on external sources such as the web; or they are based on algorithms with high computational complexity to produce query reformulations for daily maintenance tasks. The technique that we selected is based on Rocchio's method, which is perhaps the most commonly used method for query reformulation [13], [14], [16]. Rocchio's method orders the terms in the top K relevant documents based on the sum of the importance of each term of the K documents relative to the corpus by using the following equation: $Rocchio = \sum_{d2R} Tf Idf(t;d)$, where R is the set of top K relevant documents in the list of retrieved results, d is a document in R, and t is a term in d. The rst component of the measure is the Term Frequency (Tf), which is the number of times the term appears in a document and which is an indicator of the importance of the term in the document compared to the rest of the terms in that document.

The second component is the Inverse Document Frequency (Idf ), which is the inverse of the number of documents in the corpus containing that term and which indicates the speci city of that term for a document containing it.

In our work, Rocchio's method serves to expand one of the participants' search queries (i.e., base query) with the top E terms of the other participants' search queries. From the N participants' search queries that are selected for collaboration (as described in Section 4), the search query that has the highest score according to the selected criterion is set as the base query. The other participants' search queries (N-1) are set as relevant documents, whose terms are ordered, and the top E terms are used for query expansion.

For example, three participants are selected for collaboration in feature location, and the feature description that has the highest score (i.e., base query) is made up of the homogenized terms of the previous example: "breaker, convert, failur, hvac, convert, chang". The other two feature descriptions are set as relevant documents and they have the following homogenized terms: "current, convert, hvac, coverag, overload, failur, convert, assign" from the feature description "Passing of current from one converter to the HVAC assigned to its peer for coverage in case of overload or failure of the rst converter"; and the homogenized terms " failur, overload, convert, energi, air condit, unit, circuit, breaker, energi, convert, provid, provid" from the feature description "In case of failure or overload in the converter that provides energy to the air conditioning unit, the circuit breaker provides energy from its converter". By ordering the terms of the relevant documents (from highest to lowest relevance) the result is "energi, provid, current, coverag, overload, assign, overload, air, condit, unit, circuit, convert, failur, hvac, breaker". Afterwards, the base query is reformulated by adding the top ve terms of the relevant documents: "breaker, convert, failur, hvac, convert, chang, energi, provid, current, coverag, overload"

### 5.3 Fragment Retrieval on Models

Once the reformulated query is obtained and the text of the models is homogenized, we rely on an Evolutionary Algorithm [15], [16] that iterates over model fragments, modifying them using genetic operations. We have chosen to use an evolutionary algorithm because they have obtained good results by addressing similar problems with large search spaces [20]. The output of the algorithm is a model fragment ranking for the input query (requirement in TLR, bug in BL, and feature in FL).

Step 1) Initialization of model fragments. This step randomly generates an initial model fragment population from the product models, which serves as input for the evolutionary algorithm.

Step 2) Genetic Operations. This step generates a set of model fragments that could realize the reformulated query provided. The generation of new model fragments is done by applying two genetic operators that are adapted to work on model fragments: crossover and mutation [15], [16].

> The crossover operation enables the creation of a new individual by combining the genetic material from two parent model fragments. The operation looks for the model fragment of Parent 1 in the whole model of Parent 2. If the model fragment is found in the model of Parent 2, the process creates a new individual with the model fragment taken from Parent 1, but referencing the model from Parent 2. Thus, this operation enables the search space to be expanded to a different model, i.e., both model fragments (the one from Parent 1 and the one from the new individual) will be the same. However, since each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations. If the comparison does not nd the model fragment in Parent 2, the crossover returns Parent 1 (the model fragment) unchanged.
>
> The mutation operator is used to imitate the mutations that randomly occur in nature when new individuals are born. Speci cally, the mutation operator is applied to add or remove elements of the model fragment. If the operator is applied to add, one element that is directly related to elements of the model fragment is chosen to be added to the model fragment. If the operator is applied to remove, an element of the fragment is removed from the fragment. The resulting model fragment is a new candidate in the population for the realization of the input reformulated query.

Step 3) The Fitness Function. This step assesses the relevance of each of the candidate model fragments produced by ranking them according to a tness function. The objective of the tness function is the similitude between the model fragment and the reformulated query. To do this, we apply methods that are based on Information Retrieval (IR) techniques. Speci cally, we apply Latent Semantic Indexing (LSI) [39], [40] to analyze the relationships between the model fragments in the population and the reformulated query.

LSI constructs vector representations of a query and a corpus of text documents by encoding them as a term-by-document co-occurrence matrix where each row corresponds to terms and each column corresponds to documents followed by the reformulated query in the last column. Each cell of the matrix contains the number of occurrences of a term inside a document or inside the reformulated query.

In our work, the terms are all of the individual terms that are extracted from the homogenized NL of model fragments and the reformulated query, the documents are the NL representations of model fragments, and the query is the reformulated query.

Afterwards, the matrix is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [39]. One vector that represents the latent semantics of the NL texts is obtained for each document and for the reformulated query.

Finally, the similarities between each document and the reformulated query are calculated as the cosine between both of their vectors, obtaining values between -1 and 1. The tness function is as follows: $fitness(d_1) = \cos(\ ) = \frac{A \cdot B}{\|A\| \|B\|}$ where $d_1$ is a document, A is the vector representing the latent semantic of $d_1$, B is the vector representing the latent semantics of the reformulated query, and the angle

formed by the vectors A and B is .

After the similitude scores are obtained, if the stop condition is not yet met, the evolutionary algorithm will keep iterating. Once the stop condition is met, a ranking of model fragments is obtained as result. The software engineers can choose one of the model fragments of the ranking, or they can consider these solutions as a starting point from where solutions can be manually re ned, or they may re ne the query to automatically obtain different solutions.

Note that the focus of this work is on how to improve real-world criteria for selecting software engineers for collaboration. We do not make claims related to search-based approaches vs. other approaches. We think the problem is relevant when the reformulated query is used as input by search-based or other approaches.

## 6 Evaluation

### 6.1 Research questions

To address the evaluation, we formulated the following research questions:

$RQ_1$: What is the quality of the retrieved model fragment using the different criteria and the baseline in TLR, BL, and FL?

$RQ_2$: Is the difference in the quality of the solution between the different criteria and the baseline signi cant?

$RQ_3$: How much is the quality of the solution in uenced using each criterion?

### 6.2 Planning and execution

The process that was planned to answer the research questions is as follows. To start with, the data set provided by our industrial partner was taken as input. Our industrial partner, CAF, is an international provider of railway solutions all over the world. Their railway solutions can be seen in different types of trains (regular trains, subway, light rail, monorail, etc.). The data set is made up of 23 trains where product models are speci ed using a DSL for Train Control and Management, which conforms to MOF (the OMG metalanguage for de ning modeling languages that is widely used in the modelling community). The industrial supplier uses the product models to generate the rmware that controls their trains. Product models have over 27600 model elements and about 414000 properties. Each product model on average is composed of more than 1200 elements. Speci cally, the industrial partner provided the following documentation of their railway solutions: 50 requirements for TLR, 42 bug descriptions for BL, and 43 feature names for FL; the 23 models where the model fragments should be located; and the model fragment that corresponds to each requirement, bug, and feature, which will be considered to be the ground truth (oracle). The oracle was randomly extracted from documented examples from the company. They were solutions accepted by the company that have been present in their software for years. The oracle has 135 model fragments where each model fragment has from 5 to 42 model elements.

Nineteen software engineers were randomly selected from 42 software engineers who belong to three geographically distributed teams (in different cities of Spain: Zaragoza, Beasain, and Bizkaia) of the industrial partner. The selected engineers have been working from 1 to 15 years (mean of 6.65 years) for an average of 3.68 hours per day developing software. Each software engineer provided input information (search query, owner team, self-rated con dence and latest modi cation) for each requirement, feature name and bug description, as described in Section 3. The engineers' input information was used to select the participants for collaboration by following a criterion, as presented in Section 4. The result of applying each criterion is a set of the search queries of the engineers who participated in the collaboration. Afterwards, the participants' search queries were used to perform collaborative fragment retrieval on models, as described in Section 5. As a result, a model fragment was obtained for each criterion and for each requirement, feature name, and bug description.

For perspective, we compared our work with a baseline to study the impact on the results of selecting participants for collaboration. The baseline does not select participants for collaboration. The baseline takes an engineer's search query as input, and it locates the model fragment that realizes the search query using NLP and fragment retrieval on models, as described in Subsection 5.1 and Subsection 5.3, respectively. For each requirement, bug, and feature, the retrieval is performed using the engineer's search query with the highest con dence level. We chose those engineers with the highest con dence level since the industrial partner states that these engineers are supposed to achieve the best results in a solo scenario.

#### 6.2.1 Answering RQ 1

To assess what the quality of the retrieved model fragment is using the different criteria and the baseline in TLR, BL, and FL, we executed 30 independent runs for each requirement, bug, feature, criterion (four), and the baseline as suggested by [41] (i.e., 50 (requirements) x 5 (four criteria and the baseline) x 30 repetitions + 42 (bug descriptions) x 5 (four criteria and the baseline) x 30 repetitions + 43 (feature names) x 5 (four criteria and the baseline) x 30 repetitions = 20250 independent runs).

To assess the quality of each retrieved model fragment, a comparison was performed between the best retrieved model fragment of the ranking (i.e., the model fragment at position 1) and the oracle in order to calculate a confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classi cation model on a set of test data (the best solutions) for which the true values are known (from the oracle). In our case, each solution obtained is a model fragment composed of a subset of the model elements that are part of the product model. Since the granularity is at the level of model elements, the presence or absence of each model element is considered to be a classi cation. The confusion matrix distinguishes between the predicted values and the real values, classifying them into four categories: 1) True Positive (TP), values that are predicted as true (in the solution) and are true in the real scenario (the oracle); 2) False Positive (FP), values that are predicted as true (in the solution) but are false in the real scenario (the oracle); 3) True Negative (TN), values that are predicted as false (in the solution) and are false in the real scenario (the oracle); and 4) False Negative (FN), values that

are predicted as false (in the solution) but are true in the real scenario (the oracle).

From the comparison, we obtained a report with the following performance measures, which are widely accepted in the software engineering research community [17]: recall = $\frac{TP}{TP+FN}$ (measures the number of elements of the oracle that are correctly retrieved), precision = $\frac{TP}{TP+FP}$ (measures the number of elements from the solution that are correct according to the oracle), and F-measure = $2\ \frac{Precision\ \ Recall}{Precision\ +Recall}$ (corresponds to the harmonic mean of precision and recall). Recall and precision values can range from 0% to 100%. A value of 100% precision and 100% recall implies that both the solution and the oracle are the same.

#### 6.2.2 Answering RQ 2

To determine whether the difference in the quality of the solution between the different criteria and the baseline is signi cant in TLR, BL, and FL, the results should be properly compared. To do this, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [42]. The goal of our statistical analysis is to provide formal and quantitative evidence (statistical signi cance) that the criteria and the baseline do in fact have an impact on the comparison metrics (i.e., the differences were not obtained by mere chance).

The statistical tests provide a probability value, p-value, which obtains values between 0 and 1. The lower the p-value of a test, the more likely that the null hypothesis $H_0$ (de ned to state that there is no difference among the criteria and the baseline) is false. Since it is accepted by the research community that a p-value under 0.05 is statistically signi cant [42], $H_0$ can be considered false.

The test to follow depends on the properties of the data. Since our data does not follow a normal distribution, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data; however, it has been shown that the Quade test is more powerful when working with real data [43]. In addition, the Quade test has provided better results than the others when the number of algorithms is low (no more than 4 or 5 algorithms) [44],

To determine whether a criterion has a signi cant impact in the quality of the solution, the quality of the solution of the criterion should be statistically compared against all others criteria. In order to do this, we performed an additional post-hoc analysis (pair-wise comparison among criteria, which also includes the baseline).

#### 6.2.3 Answering RQ 3

To determine how much the quality of the solution is in uenced using each criterion, it is important to assess (through effect size measures) if a criterion is statistically better than another one, and if so, the magnitude of the improvement.

For a non-parametric effect size measure, we used Vargha and Delaney's $\hat{A}_{12}$ [45], [46]. $\hat{A}_{12}$ measures the probability that running one criterion yields higher values than running another criterion. If the two criteria are equivalent, then $\hat{A}_{12}$ will be 0:5. For example, $\hat{A}_{12} = 0:7$ between Criterion 1 and Criterion 2 means that Criterion 1 would obtain better results in 70% of the runs, and $\hat{A}_{12} = 0:3$ means that Criterion 2 would obtain better results in 70% of the runs. We recorded an $\hat{A}_{12}$ value for every pair of criteria as well as for every criterion and the baseline in TLR, BL, and FL.

### 6.3 Implementation details

We implemented the selection of participants for collaboration using Java. The number of the engineers to be selected for collaboration was set to four (one engineer provided the base query and three engineers provided relevant queries to reformulate) and we considered the rst 10 term suggestions to expand the base query. We principally chose these values by following the recommendation of the domain literature [15], [16], [37].

We used the Eclipse Modeling Framework to manipulate the models and to manage the model fragments. To implement the techniques that support Natural Language Processing, we used OpenNLP [47] for the POS-Tagger and the English (Porter2) stemming algorithm [48] for the stemming algorithm (originally created using snowball and then compiled to Java). The LSI was implemented using the Ef cient Java Matrix Library (EJML [49]).

The genetic operations are built upon the Watchmaker Framework for Evolutionary Computation [50]. The con guration parameters for the algorithm are as follows: the number of generations (i.e., repetitions of the genetic operations and tness loop) is 2500 since it is the value needed by our case study to converge, the size of the population is 100, the number of parents is 2, the number of offspring from parents is 2, the crossover probability is 0.9, and the mutation probability is 0.1. For those settings, we chose values that are commonly used in the Model Fragment Retrieval literature [15], [16].

We are limited by the con dentiality agreements that we have with the industrial partner. The implementation and the data are not available. Implementation of Collaborative Fragment Retrieval is currently being used by the industrial partner. The trains of the data set are currently operating and under maintenance contracts, or will be released in the near future. Nevertheless, for purposes of replicability, the csv les used as input in the statistical analysis are available at: https://svit.usj.es/criteria-for-collaboration/.

## 7 Results

### 7.1 Research Question 1

Table 1 shows the mean values and standard deviations of recall, precision, and F-measure for the 50 requirements (TLR), the 42 bugs (BL), and the 43 features (FL) of the industrial case study for the four criteria and the baseline.

$RQ_1$ answer. The results revealed that the criterion that obtained the best result is different in TLR, BL, and FL. In TLR, Criterion 2 (most con dent) obtained the best result in terms of recall, precision, and F-measure (89.08%, 90.59%, and 89.58%, respectively). In BL, Criterion 3 (latest modi cations) obtained the best result in terms of recall, precision, and F-measure (72.31%, 66.34%, and 68.01%, respectively). In FL, Gold Criterion (most con dent and latest modi cation owners) obtained the best result, providing an average value of 91.31% in recall, 92.69% in precision, and 91.83% in F-measure.

TABLE 1
Mean Values and Standard Deviations for Recall, Precision, and F-Measure in the industrial case study

| | Recall ( ) | | | | | |
|---|---|---|---|---|---|---|
| | TLR | | BL | | FL | |
| Criterion 1 | 70.58 | 15.78 | 46.10 | 13.60 | 69.22 | 12.08 |
| Criterion 2 | 89.08 | 6.26 | 40.93 | 16.27 | 90.07 | 6.76 |
| Criterion 3 | 53.16 | 15.28 | 72.31 | 13.92 | 67.58 | 14.59 |
| Gold criterion | 68.50 | 14.33 | 64.47 | 15.53 | 91.31 | 6.52 |
| Baseline | 48.15 | 15.08 | 35.95 | 14.49 | 65.83 | 14.99 |
| | Precision ( ) | | | | | |
| | TLR | | BL | | FL | |
| Criterion 1 | 71.08 | 16.74 | 35.72 | 17.45 | 77.52 | 14.16 |
| Criterion 2 | 90.59 | 7.24 | 33.99 | 17.41 | 92.06 | 5.76 |
| Criterion 3 | 56.83 | 13.30 | 66.34 | 13.71 | 72.86 | 12.70 |
| Gold criterion | 69.70 | 9.60 | 58.84 | 14.92 | 92.69 | 4.36 |
| Baseline | 51.96 | 14.61 | 28.12 | 15.45 | 68.80 | 13.86 |
| | F-measure ( ) | | | | | |
| | TLR | | BL | | FL | |
| Criterion 1 | 68.76 | 11.54 | 36.39 | 13.63 | 71.81 | 9.42 |
| Criterion 2 | 89.58 | 4.92 | 33.19 | 13.00 | 90.84 | 4.62 |
| Criterion 3 | 53.07 | 11.64 | 68.01 | 10.75 | 68.90 | 9.98 |
| Gold criterion | 68.10 | 9.48 | 59.17 | 11.17 | 91.83 | 4.09 |
| Baseline | 47.74 | 10.97 | 27.55 | 12.13 | 66.06 | 11.11 |

### 7.2 Research Question 2

The p-V alues obtained in the Quade test were lower than 0.05 in all cases, so we reject the null hypothesis. Consequently, we can state that there are signi cant differences among the criteria and the baseline in TLR, BL, and FL for all the performance indicators.

The upper part of Table 2 shows the p-V alues of Holm's post-hoc analysis for pair-wise comparison of criteria and the baseline in the performance indicators of TLR, BL, and FL. The majority of the p-V alues are lower than their corresponding signi cance threshold value (0.05), indicating that the differences in performance using the criteria are signi cant. However, some values are greater than the threshold, indicating that the differences between those pair-wise comparisons are not signi cant.

$RQ_2$ answer. From the results, we conclude that the criteria and the baseline have signi cant differences in TLR, BL, and FL. In TLR, the F-measure shows that all criteria (Criterion 1, Criterion 2, Criterion 3, and Gold criterion) produce a signi cant improvement compared to the baseline. In BL, the F-measure shows that all of the criteria except Criterion 2 produce a signi cant improvement in the quality of the solution with regard to the baseline. In FL, the F-measure shows that all of the criteria except Criterion 3 produce a signi cant improvement compared to the baseline.

### 7.3 Research Question 3

The lower part of Table 2 shows the values of the effect size statistics between pair-wise comparisons in TLR, BL, and FL. In TLR, the largest differences were obtained in comparisons that entail Criterion 2, where the largest difference is obtained when compared with the baseline (0.996 for recall, 0.9968 for precision, and 1 for F-measure). Therefore, in TLR, Criterion 2 outperforms the baseline with a pronounced superiority (99.6% of the times for recall, 99.68% of the times for precision, and 100% of the times for F-measure). In BL, Criterion 3 obtains the largest differences when compared with the baseline. Criterion 3 outperforms the baseline with a pronounced superiority (96.32% of the times for recall, 96.2% of the times for precision, and 98.92% of the times for F-measure). In FL, Criterion 2 and Gold criterion show a pronounced superiority over Criterion 1, Criterion 3, and the baseline. The largest difference is obtained when comparing Criterion 3 and Gold criterion (0.0857 for recall, 0.0776 for precision, and 0.0092 for F-measure). Therefore, Gold criterion outperforms Criterion 3 with a pronounced superiority (91.43% of the times for recall, 92.24% of the times for precision, and 99.08% of the times for F-measure).

$RQ_3$ answer. From the results, we can conclude how much the quality of the solution was in uenced using each criterion in TLR, BL, and FL (see the lower part of Table 2).

## 8 Discussion and Lessons Learned

After analyzing the results, we present the following recommendations for TLR, BL, and FL. For TLR, the results reveal that collaboration should avoid involving software engineers that are only from the owner team. This is because part of the domain knowledge related to the requirement is often assumed and not embodied when search queries are written by the members of the owner team. For example, given the requirement: *At all stations, the doors are automatically opened*, the engineers understand that the doors have to be opened in all of the stations without being requested by a passenger. However, this requirement also embodies tacit knowledge that is not written but that is obvious to the owner engineers: The train has doors on both sides, but only the doors on the side of the platform will be opened while the doors on the side of the tracks will remain closed, and all the doors of one side will be opened, except the driver's door in the cabin.

A previous work [51] shows differences in style between requirements that are written by different teams in a company. Given a requirement, every software engineer of the company can easily determine whether or not the requirement belongs to his/her team. However, our collaborative model maintenance experience revealed a surprising turn. When confronting non-familiar requirements, a software engineer produces longer search queries with less implicit knowledge. A rst glance, unfamiliarity to the requirement may be seen as a disadvantage to producing a search query, but this unfamiliarity also drives the software engineer to produce a detailed search query.

Since the model fragment location depends on the domain knowledge that is encoded in the words of the search query, the location takes advantage of the explicit information that the engineer from a non-owner team provides. Therefore, the tacit knowledge issue can be mitigated with collaboration by involving a software engineer from a non-owner team.

However, involving engineers from different teams also entails disadvantages because each team develops its own in-house terms. This contributes to a vocabulary mismatch issue (i.e., one concept is speci ed using different terms). If the terms that are used in the requirements and the terms that are used in the models are not known synonyms, they cannot be related, and, therefore, the requirement cannot be correctly related to the elements of the model. Therefore, the

TABLE 2
Holm's post hoc p-V alues and $\hat{A}_{12}$ statistic for each pair

Holm's post hoc p-V alues

| | TLR | | | BL | | | FL | | |
|---|---|---|---|---|---|---|---|---|---|
| | Recall | Precision | F-measure | Recall | Precision | F-measure | Recall | Precision | F-measure |
| C1 vs C2 | $5{:}5\text{x}10^{-11}$ | $4{:}2\text{x}10^{-10}$ | $7\text{x}10^{-15}$ | 0.024 | 0.45 | 0.19 | $8{:}4\text{x}10^{-13}$ | $3{:}7\text{x}10^{-9}$ | $2\text{x}10^{-14}$ |
| C1 vs C3 | $7{:}7\text{x}10^{-8}$ | $5{:}1\text{x}10^{-6}$ | $5{:}2\text{x}10^{-10}$ | $7{:}3\text{x}10^{-11}$ | $2{:}1\text{x}10^{-13}$ | $4{:}1\text{x}10^{-14}$ | 0.67 | 0.07 | 0.23 |
| C1 vs Gold | 0.62 | 0.76 | 0.68 | $1\text{x}10^{-1}$ | $1{:}2\text{x}10^{-1}$ | $1{:}5\text{x}10^{-10}$ | $1{:}3\text{x}10^{-13}$ | $3\text{x}10^{-10}$ | $2\text{x}10^{-14}$ |
| C1 vs Baseline | $1{:}7\text{x}10^{-9}$ | $8{:}4\text{x}10^{-8}$ | $3{:}2\text{x}10^{-12}$ | 0.0024 | 0.068 | 0.0016 | 0.32 | 0.004 | 0.021 |
| C2 vs C3 | $2\text{x}10^{-16}$ | $2\text{x}10^{-16}$ | $2\text{x}10^{-16}$ | $1{:}6\text{x}10^{-11}$ | $3{:}5\text{x}10^{-13}$ | $3{:}1\text{x}10^{-14}$ | $8{:}6\text{x}10^{-12}$ | $3{:}9\text{x}10^{-11}$ | $5{:}9\text{x}10^{-14}$ |
| C2 vs Gold | $3{:}7\text{x}10^{-12}$ | $3{:}6\text{x}10^{-16}$ | $3{:}6\text{x}10^{-16}$ | $3\text{x}10^{-10}$ | $2{:}5\text{x}10^{-7}$ | $1{:}5\text{x}10^{-12}$ | 0.35 | 0.53 | 0.26 |
| C2 vs Baseline | $2\text{x}10^{-16}$ | $2\text{x}10^{-16}$ | $2\text{x}10^{-16}$ | 0.18 | 0.36 | 0.06 | $2{:}6\text{x}10^{-13}$ | $1{:}3\text{x}10^{-12}$ | $1{:}5\text{x}10^{-14}$ |
| C3 vs Gold | $7{:}3\text{x}10^{-7}$ | $4\text{x}10^{-7}$ | $6{:}9\text{x}10^{-9}$ | 0.036 | 0.028 | 0.0016 | $8{:}6\text{x}10^{-12}$ | $2{:}6\text{x}10^{-13}$ | $2\text{x}10^{-14}$ |
| C3 vs Baseline | 0.13 | 0.12 | 0.034 | $7{:}1\text{x}10^{-14}$ | $3{:}1\text{x}10^{-14}$ | $3{:}1\text{x}10^{-14}$ | 0.48 | 0.2 | 0.15 |
| Gold vs Baseline | $3{:}3\text{x}10^{-10}$ | $1\text{x}10^{-10}$ | $8{:}5\text{x}10^{-15}$ | $2{:}3\text{x}10^{-11}$ | $8{:}5\text{x}10^{-12}$ | $9{:}4\text{x}10^{-14}$ | $3{:}8\text{x}10^{-12}$ | $7{:}6\text{x}10^{-14}$ | $1{:}3\text{x}10^{-13}$ |

$\hat{A}_{12}$ statistic

| | TLR | | | BL | | | FL | | |
|---|---|---|---|---|---|---|---|---|---|
| | Recall | Precision | F-measure | Recall | Precision | F-measure | Recall | Precision | F-measure |
| C1 vs C2 | 0.144 | 0.1574 | 0.048 | 0.6060 | 0.5368 | 0.5692 | 0.0719 | 0.1650 | 0.0454 |
| C1 vs C3 | 0.7812 | 0.7328 | 0.8336 | 0.0811 | 0.0816 | 0.0266 | 0.5376 | 0.6161 | 0.5695 |
| C1 vs Gold | 0.5478 | 0.524 | 0.506 | 0.1842 | 0.1593 | 0.0941 | 0.0614 | 0.1366 | 0.0281 |
| C1 vs Baseline | 0.842 | 0.7904 | 0.9084 | 0.6718 | 0.6378 | 0.6995 | 0.5911 | 0.6836 | 0.6593 |
| C2 vs C3 | 0.978 | 0.9948 | 0.9988 | 0.0692 | 0.0771 | 0.0153 | 0.9051 | 0.9135 | 0.9832 |
| C2 vs Gold | 0.8968 | 0.9604 | 0.9788 | 0.1451 | 0.1440 | 0.0646 | 0.4448 | 0.5059 | 0.4299 |
| C2 vs Baseline | 0.996 | 0.9968 | 1 | 0.5760 | 0.6003 | 0.6332 | 0.9048 | 0.9221 | 0.9773 |
| C3 vs Gold | 0.2184 | 0.2044 | 0.1464 | 0.6531 | 0.6145 | 0.7177 | 0.0857 | 0.0776 | 0.0092 |
| C3 vs Baseline | 0.5944 | 0.5952 | 0.6372 | 0.9632 | 0.9620 | 0.9892 | 0.5473 | 0.5916 | 0.5917 |
| Gold vs Baseline | 0.8312 | 0.8484 | 0.9252 | 0.9167 | 0.9155 | 0.9626 | 0.9140 | 0.9259 | 0.9751 |

lack of awareness that is caused by the vocabulary mismatch makes it impossible to locate the elements from the model that are relevant to the requirement. To address this issue, it is necessary to extend the NLP techniques with a thesaurus that contains the in-house terms of the different teams.

For BL, the results show that collaboration should avoid involving software engineers that only take into account high con dence levels. A high con dence level suggests that the software engineer has a deep understanding of the functionality that is intuitively related to the bug. However, this understanding is not always enough. In most of the cases, bugs were connected to recent modi cations to the models.

A high con dence level and the participation in recent modi cations sounds like the perfect pro le for collaborating in BL. However, a low con dence level and the participation in recent modi cations was also relevant for a signi cant number of cases. Therefore, we can only state that participating in recent modi cations should be specially considered for collaboration in the context of BL. This recommendation is aligned with the nding of the Defect Principle, which states that the most recent modi cations of a project are the most relevant for certain Information Retrieval purposes [18].

For FL, Gold criterion does not achieve perfect values because achieving the maximum number of model elements takes into account the involvement of software engineers with low con dence levels. For example, in the case of locating a feature related to the braking equipment, a software engineer with expertise in the train coupling (i.e., two trains are physically connected and only one of them commands the resulting train unit) declares a low con dence level in his search query because he is not an expert on the braking equipment, but his query describes what happens to the braking when two trains are coupled. This information is not produced by an expert on the braking system who declares a high con dence level.

Our analysis of the results reveals that the con dence level is not powerful enough to assess the software engineers' participation in FL. Engineers with information that is hard to come by which describes a small part of the feature declare themselves as low con dence level. Therefore, we should also ask software engineers about the percentage of coverage that they think their search query may achieve, and, consequently, the con dence level for that coverage.

### 8.1 Focus group interview

We ran a focus group to obtain qualitative data from the 19 selected software engineers of our industrial partner. Speci cally, the focus group was composed of the following open questions: (1) What do you think about the criteria for selecting participants for collaboration?; (2) What do you think about the results of each criterion?; and (3) Why would you choose the results of one criterion over the results of the baseline?

The engineers stated that the criteria were appropriate and complete according to their daily maintenance tasks. There was a consensus among the engineers that the Gold criterion (the most con dent owners who performed the latest modi cation) should get the best results in all maintenance tasks.

After checking the results, the engineers highlighted that they did not expect the Gold criterion to not obtain the best results in all maintenance activities. They found the results to be counterintuitive because they thought that those engineers who are not con dent (i.e., underdogs) should not be

involved in the collaboration. However, the engineers realized that the results improved because underdogs produced longer queries with details that helped to obtain a model fragment that was more complete than the model fragment retrieved by con dent engineers, who omitted details in the queries because they were considered to be obvious.

The engineers also acknowledged the importance of collaboration during maintenance tasks after checking the results. The engineers mentioned that they would choose the results of a criterion (collaboration) instead of the results of the baseline (without collaboration) since they stated that is dif cult to have full knowledge while maintenance tasks are being performed. Moreover, the engineers highlighted that this work indicates that they were missing the potential knowledge of underdogs to obtain better results.

## 9 Threats to validity

To acknowledge the limitations of our evaluation, we use the classi cation of threats of validity of [52], [53], which distinguishes four aspects of validity.

Construct Validity: Our evaluation uses three measures to minimize this risk: precision, recall, and F-measure, which are widely accepted in the software engineering research community [17].

Internal Validity: We used an oracle (obtained from our industrial partner, which is considered the ground truth) where the expected solution was known beforehand. By doing so, we were able to evaluate the different criteria for the selection of participants for collaboration in TLR, BL, and FL and to compute the recall, precision, and F-measure. Another threat of this type is poor parameter settings. In this paper, we used values that are commonly used in the Model Fragment Retrieval literature [15], [16]. For the number of relevant documents and terms used to expand the query, we used the values of 3 and 10, respectively, as recommended in the literature [15], [16], [37]. As suggested by Arcuri and Fraser [41], default values are good enough to measure the performance. However, at this stage, we do not know how using different values would impact the results.

External Validity: To mitigate this threat, our work has been designed not only to be applied to the domain of the industrial partner but also to different domains. The requisites to apply our work are that the set of models where requirement, bugs, or features have to be located must conform to MOF (the OMG metalanguage for de ning modeling languages), the queries must be provided as a textual description in natural language, and the engineers' input information must be provided (owner team, self-rated con dence, and latest modi cation). We think that natural language queries and MOF-based models would apply in a wide variety of model driven engineering scenarios.

As occurs in other works [54], [55], [16], the results depend on the quality of the queries. It is also worth noting that the language used for the textual elements of the models and the feature descriptions in the query provided must be the same. This language is speci c to each domain.

Hence, even though our approach can be applied to locate requirements, bugs, and features on MOF-based models from different domains, our approach should be applied to other domains before assuring its generalization.

Reliability: To reduce this threat, even though the industrial partner provided the input information (the requirement, bug and feature descriptions, engineers' input information, and the product models) they were not involved in this research.

## 10 Related work

Previous works spent their efforts on improving the systems that could boost an individual's search effectiveness by addressing collaborative information retrieval [56], [57]. Yue et al. [57] developed a web search system to investigate factors that in uence query reformulation in the context of explicit Collaborative Information Retrieval based on user analysis of human subjects. Query reformulation can be automatically performed to add terms that are either similar or related to a user query [58]. Most existing research is focused on query expansion by nding terms in relevant documents such as source code and Internet sites [2], [59], [3], [4], [60], [5], [6], [55]. Sirres et al. [59] propose a technique for nding relevant code using free-form query terms from internet sites such as Q&A posts from Stack Over ow. Hill et al. [55] extract possible query expansion terms from the code using word context. Cao et al. [2] propose an automated query reformulation approach for ef cient search using query logs provided by Stack Over ow.

In contrast, other approaches propose automatically reformulating the query by removing words to reduce long queries. Chaparro et al. [61] reduce terms of a low-quality query to only include the terms describing the Observed behaviour (OB), which describes the current (mis)behaviour (i.e., incorrect or unexpected behaviour) of software. Chaparro et al. [62] evaluate a set of query reformulation strategies using existing information in bug descriptions and the removal of irrelevant parts from the original query. Kumaran and Carvalho [63] analyze the most promising subsets of terms from the original query to reduce queries. Haiduc et al. [14] propose an approach that is trained with a sample of queries and relevant results in order to automatically recommend an automatic query reformulation technique (expansion or reduction) to improve the performance. Florez et al. [64] combine query reduction and expansion techniques to improve the effectiveness of bug localization. Other works have been proposed to improve the effectiveness of feature location by involving users' feedback about the relevance of the retrieved results. For example, Wang et al. [65] propose a code search approach, which incorporates user feedback to re ne the query. Despite the effort to improve the performance of retrieving code by automatically reformulating the queries, it has been neglected in models and in industry.

Fig. 3 shows the connections and differences between our previously published works and this work. At the bottom of the gure, it is possible to nd our work regarding Search-Based Software Engineering (SBSE) [66]. While [66] also uses automatic query reformulations, its goal is completely different since it does not address collaboration in SBSE. It uses automatic query reformulations as operations of the evolutionary algorithm instead of using the widespread single-point crossover plus random mutation,

leveraging the latent semantics that models hold rather than randomly generating new candidate solutions.

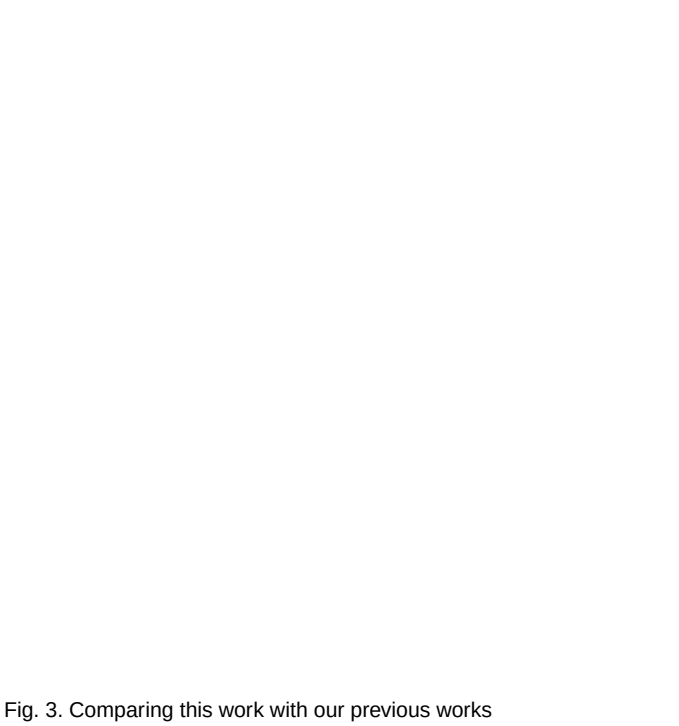

Fig. 3. Comparing this work with our previous works

The middle part of Fig. 3 shows the work in [67], where the human plays the role of the tness function of the evolutionary algorithm. As one of its outcomes, the work in [67] motivates the need for collaboration in order to share the burden of evaluating candidate solutions, which could lead to success in problems where a single human fails.

The upper part of Fig. 3 shows our previously published works addressing collaboration and their differences with this work. Moreover, Table 3 compares these works with regard to several factors, namely: the number of queries that the different works evaluate, the criteria for collaboration that they use, the reformulation techniques that are applied, and the software engineering tasks that are addressed. In [68], an engineer's query is enriched (adding/removing terms) using an automatic query reformulation technique, which takes terms of the product models as input. This leads to negative results since the reformulated queries do not improve the performance in models. The work in [69] was our rst approach where the criterion of the most con dent engineer is applied to address collaboration among different engineers (without an evolutionary algorithm to perform the location of features). Through the work presented in [70], the work in [69] was extended with a low-cost variant, which limits the time that engineers can spend for providing knowledge. This last work also explores other existing query reformulation techniques. Finally, the work in [16] studies the impact that the number of engineers that participate in the collaboration has over the quality of the solution, and whether the inclusion of the engineers' con dence produces an improvement in the results.

In contrast to the above works, the novelty of this work puts the focus on the selection of participants for collaboration, with the aim of answering the question on who should participate in collaboration. To do this, this paper explores

TABLE 3
Comparing our works that address collaboration

| | Queries | Criteria | Reformulation techniques | Software engineering tasks |
|---|---|---|---|---|
| This work | 2565 | C1: Available owners<br>C2: The most con dent engineer<br>C3: The latest modi cation<br>C4: Gold | Rocchio | FL<br>TLR<br>BL |
| AUSE [16] | 817 | C2: The most con dent engineer | Rocchio | FL |
| Access [70] | 817 | C2: The most con dent engineer | Rocchio, RSV, Dice, Reduction | FL |
| CoopIS [69] | 817 | C2: The most con dent engineer | Rocchio | FL |
| DKE [68] | 217 | - | Rocchio, RSV, Dice, Reduction | FL |

how the quality of the results is affected by different real-world criteria for selecting participants for collaboration (Available owners, The most con dent engineer, and The latest modi cation) in different maintenance tasks (TLR, BL, and FL). This implies using the highest number of queries that we evaluated so far (as the second column of Table 3 shows). Moreover, the intuition of our industrial partner suggests that a combination of two criteria (The most con-dent engineer and The latest modi cation) should be the criterion that obtains the best results. This new criterion, identi ed as the Gold criterion, has never been used before by our industrial partner nor by our previous research, and is explored for the rst time in this paper. Finally, our work uncovers novel recommendations (even some counterintuitive ones, such as the inclusion of engineers that might be seen as not relevant) towards assembling a team of engineers for collaboration.

## 11 Conclusion

We have analyzed collaborative maintenance tasks (TLR, BL, and FL) on software models in a real-world industrial domain. This kind of real-world experience is hard to obtain since the majority of related works on collaboration use academic data. We do not claim collaboration should be systematically applied to every case. Collaboration, as in our work, is necessary when the requirement/bug/feature signi cantly transcends the knowledge of a single software engineer. We should mention we do not claim that for every requirement/bug/feature all engineers should produce a search query to collaborate. Actually, it is the opposite. Our work helps to make decisions on the selection of engineers for collaboration.

We have also compared four criteria for collaboration: three criteria for collaboration that were used indistinctly in the industry and a criterion that seemed to be the best but which, counterintuitively in most cases, has not yielded the best results. Our results show that collaboration in the maintenance of industrial models pays off. However, to release the full potential of collaboration, we should challenge our intuition in the selection of participants. The lessons learned show how to improve real-world criteria for selecting software engineers for collaboration. Therefore,

this work provides a better understanding of the profiles that work best for the three software tasks (TLR, BL, and FL), which are among the most common and relevant maintenance tasks in the Software Engineering field. Furthermore, this work raises awareness of the positive role that underdogs (software engineers with low confidence levels) can play in collaboration.

## ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project ALPS under Grant RTI2018-096411-B-I00, and in part by the Gobierno de Aragón (Spain) (Research Group S05_20D).

## REFERENCES

[1] M. Franzago, D. D. Ruscio, I. Malavolta, and H. Muccini, "Collaborative model-driven software engineering: a classification framework and a research map," IEEE Transactions on Software Engineering, vol. PP, no. 99, pp. 1–1, 2017.

[2] K. Cao, C. Chen, S. Baltes, C. Treude, and X. Chen, "Automated query reformulation for efficient search based on query logs from stack overflow," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 1273–1285.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery," in IEEE 18th International Conference on Program Comprehension, 2010.

[8] A. Mahmoud, N. Niu, and S. Xu, "A Semantic Relatedness Approach for Traceability Link Recovery," in IEEE 20th International Conference on Program Comprehension, 2012.

[9] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey," in Journal of Software Maintenance and Evolution: Research and Practice, 2011.

[10] J. Rubin and M. Chechik, "A survey of feature location techniques," in Domain Engineering Springer, 2013, pp. 29–58.

[11] B. Selic, "The pragmatics of model-driven development," IEEE Softw., vol. 20, no. 5, pp. 19–25, Sep. 2003. [Online]. Available: http://dx.doi.org/10.1109/MS.2003.1231146

[12] M. Brambilla, J. Cabot, and M. Wimmer, Model-Driven Software Engineering in Practice, 1st ed. Morgan & Claypool Publishers, 2012.

[13] L. Martie, T. D. LaToza, and A. van der Hoek, "Codeexchange: Supporting reformulation of internet-scale code queries in context," in 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, pp. 24–35. [Online]. Available: https://doi.org/10.1109/ASE.2015.51

[14] 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.

[15] F. Pérez, R. Lapeña, J. Font, and C. Cetina, "Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study," Information & Software Technology, vol. 103, pp. 188–201, 2018. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.06.017

[16] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," Automated Software Engineering, vol. 26, no. 1, pp. 161–202, 2019. [Online]. Available: https://doi.org/10.1007/s10515-019-00251-9

[17] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval New York, NY, USA: McGraw-Hill, Inc., 1986.

[18] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," in Proceedings of the 26th International Conference on Software Engineering, 2004.

[19] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "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, ser. MODELS '16. New York, NY, USA: ACM, 2016, pp. 272–282. [Online]. Available: http://doi.acm.org/10.1145/2976767.2976789

[20] ——, "Achieving feature location in families of models through the use of search-based software engineering," IEEE Transactions on Evolutionary Computation, vol. PP, no. 99, pp. 1–1, 2017.

[21] M. Harman and B. F. Jones, "Search-based software engineering," Information & Software Technology, vol. 43, no. 14, pp. 833–839, 2001. [Online]. Available: https://doi.org/10.1016/S0950-5849(01)00189-6

[22] A. E. Hassan and R. C. Holt, "The Top Ten List: Dynamic Fault Prediction," in 21st IEEE International Conference on Software Maintenance, 2005.

[23] B. Sisman and A. C. Kak, "Incorporating Version Histories in Information Retrieval Based Bug Localization," in 9th IEEE Working Conference on Mining Software Repositories, 2012.

[24] M. R. Karim, G. Ruhe, M. M. Rahman, V. Garousi, and T. Zimmermann, "An empirical investigation of single-objective and multi-objective evolutionary algorithms for developer's assignment to bugs," Journal of Software: Evolution and Process, vol. 28, no. 12, pp. 1025–1060, 2016.

[25] M. M. Rahman, M. R. Karim, G. Ruhe, V. Garousi, and T. Zimmermann, "An empirical investigation of a genetic algorithm for developer's assignment to bugs," in Proceedings of the First North American Search Based Software Engineering Symposium, February 2015.

[26] M. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, October 15-16, 2009, Lake Buena Vista, Florida, USA, 2009, pp. 439–442.

[27] M. M. Rahman, S. M. Sohan, F. Maurer, and G. Ruhe, "Evaluation of optimized staffing for feature development and bug fixing," in Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010, 16-17 September 2010, Bolzano/Bozen, Italy, 2010.

[28] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""not my bug!" and other reasons for software bug report reassignments," in Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work, ser. CSCW '11. ACM, 2011, pp. 395–404.

[29] K. Kevic, S. C. Müller, T. Fritz, and H. C. Gall, "Collaborative bug triaging using textual similarities and change set analysis," in 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), May 2013, pp. 17–24.

[30] H. Kagdi and D. Poshyvanyk, "Who can help me with this change request?" in 2009 IEEE 17th International Conference on Program Comprehension (ICPC 2009), 2009.

[31] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595715

[32] K. Nakakoji, "Supporting software development as collective creative knowledge work," in 2nd International Workshop on Supporting Knowledge Collaboration in Software Development, 2006.

[33] M. Bass, J. D. Herbsleb, and C. Lescher, "Collaboration in global software projects at siemens: An experience report," in 2nd IEEE International Conference on Global Software Engineering, ICGSE 2007, Munich, Germany, 27-30 August, 2007, 2007, pp. 33–39.

[34] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture." in CSCW. ACM, 2000, pp. 231–240.

[35] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, 19-25 May 2002, Orlando, Florida, USA*, 2002, pp. 503–512.

[36] 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.

[37] 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.

[38] 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. Lämmel, and C. Verhoef, Eds. IEEE Computer Society, 2008, pp. 123–132.

[39] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse processes*, vol. 25, 1998.

[40] T. Hofmann, "Probabilistic Latent Semantic Indexing," in *Proceedings of the 22nd Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[41] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9249-9

[42] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, May 2014. [Online]. Available: http://dx.doi.org/10.1002/stvr.1486

[43] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2009.12.010

[44] W. Conover, *Practical nonparametric statistics*, 3rd ed., ser. Wiley series in probability and statistics. New York, NY [u.a.]: Wiley, 1999.

[45] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000. [Online]. Available: http://jeb.sagepub.com/content/25/2/101.abstract

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

[47] "Apache opennlp: Toolkit for the processing of natural language text," https://opennlp.apache.org/, 2021.

[48] "English (porter2) stemming algorithm," http://snowball.tartarus.org/algorithms/english/stemmer.html, 2021.

[49] "Efficient java matrix library," http://ejml.org/, 2021.

[50] D. Dyer, "The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java)," http://watchmaker.uncommons.org/, 2016, [Online; accessed 7-April-2016].

[51] J. Echeverría, F. Pérez, J. I. Panach, C. Cetina, and O. Pastor, "The influence of requirements in software model development in an industrial environment," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM*, 2017.

[52] 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.

[53] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, 2012.

[54] 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*, 2013, pp. 309–318.

[55] 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. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2009.5070524

[56] C. Shah and R. González-Ibáñez, "Evaluating the synergic effect of collaboration in information seeking," in *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '11. New York, NY, USA: ACM, 2011, pp. 913–922. [Online]. Available: http://doi.acm.org/10.1145/2009916.2010038

[57] Z. Yue, S. Han, D. He, and J. Jiang, "Influences on query reformulation in collaborative web search," *Computer*, vol. 47, no. 3, pp. 46–53, Mar. 2014. [Online]. Available: doi.ieeecomputersociety.org/10.1109/MC.2014.62

[58] C. Shah, "Collaborative information seeking: A literature review," *Exploring The Digital Frontier Advances In Librarianship*, vol. 32, 2010.

[59] R. Sirres, T. F. Bissyandé, D. Kim, D. Lo, J. Klein, K. Kim, and Y. L. Traon, "Augmenting and structuring user queries to support efficient free-form code search," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2622–2654, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9544-y

[60] W. Wang, A. Gupta, N. Niu, L. D. Xu, J. C. Cheng, and Z. Niu, "Automatically tracing dependability requirements via term-based relevance feedback," *IEEE Trans. Industrial Informatics*, vol. 14, no. 1, pp. 342–349, 2018.

[61] O. Chaparro, J. M. Florez, and A. Marcus, "Using observed behavior to reformulate queries during text retrieval-based bug localization," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, vol. 00, Sept. 2018, pp. 376–387. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ICSME.2017.100

[62] ——, "Using bug descriptions to reformulate queries during text-retrieval-based bug localization," *Empirical Software Engineering*, vol. 24, no. 5, pp. 2947–3007, 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9672-z

[63] 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.

[64] J. M. Florez, O. Chaparro, C. Treude, and A. Marcus, "Combining query reduction and expansion for text-retrieval-based bug localization," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 166–176.

[65] 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. [Online]. Available: http://doi.acm.org/10.1145/2642937.2642947

[66] F. Pérez, T. Ziadi, and C. Cetina, "Utilizing automatic query reformulations as genetic operations to improve feature location in software models," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, jun 2020. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TSE.2020.3000520

[67] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Empowering the human as the fitness function in search-based model-driven engineering," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, oct 2021.

[68] 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.

[69] 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. [Online]. Available: https://doi.org/10.1007/978-3-319-69462-7_9

[70] ——, "Evaluating low-cost in internal crowdsourcing for software engineering: The case of feature location in an industrial environment," *IEEE Access*, vol. 8, pp. 65 745–65 757, 2020.

**Francisca Pérez** is Associate Professor in the SVIT Research Group (https://svit.usj.es) at San Jorge University. She received a PhD in Computer Science from the Polytechnic 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-level international software engineering conferences and journals, such as IEEE Transactions on Software Engineering (TSE), 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.

**Raúl Lapeña** received a PhD in Computer Science from the Polytechnic University of Valencia. He is currently Assistant Professor in the SVIT Research Group at San Jorge University. His main research interests lie in model-driven development, feature location, and software product lines. He publishes his research results and participates in high-quality international software engineering conferences and journals, such as the Conference on Advanced Information Systems Engineering (CAiSE), the Information and Software Technology (IST) journal, and the Journal of Systems and Software (JSS).

**Ana C. Marcén** received a PhD in Computer Science from the Polytechnic University of Valencia. She is currently Assistant Professor in the SVIT Research Group at San Jorge University. She publishes her research results and participates in high-quality international software engineering conferences and journals, such as the Conceptual Modeling (MoDELS) conference, the Information and Software Technology (IST) journal, and the Journal of Systems and Software (JSS). Her current research interests include model-driven development, feature location, traceability link recovery, and machine learning.

**Carlos Cetina** is Associate Professor with San Jorge University and the Head of the SVIT Research Group. He received a PhD in Computer Science from the Polytechnic University of Valencia. His research focuses on software product lines and model-driven development. His research results have reshaped software development in world-leading 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.