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

FRANCISCA PÉREZ, RAÚL LAPEÑA, ANA C. MARCÉN, and CARLOS CETINA*,

SVIT Research Group, Universidad San Jorge, Spain

In industry, software 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 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 relies on an evolutionary algorithm. Our work uncovers how software engineers who might be seen as not being relevant in the collaboration can lead to significantly better results. A focus group confirmed the relevance of the findings.

CCS Concepts: • **Software and its engineering → Maintaining software**; **Search-based software engineering**; • **General and reference → Empirical studies**.

Additional Key Words and Phrases: Collaborative Software Engineering, Search-Based Software Engineering, Model-Driven Engineering

ACM Reference Format:

Francisca Pérez, Raúl Lapeña, Ana C. Marcén, and Carlos Cetina. 2022. How the Quality of Maintenance Tasks is Affected by Criteria for Selecting Engineers for Collaboration. *ACM Trans. Softw. Eng. Methodol.* 0, 0, Article 0 (March 2022), 22 pages. https://doi.org/XXXXXXX.XXXXXXX

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 [19].

Previous works [8, 35, 41, 51, 52] 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

*Also with University College London.

Authors' address: Francisca Pérez, mfperez@usj.es; Raúl Lapeña, rlapena@usj.es; Ana C. Marcén, acmarcen@usj.es; Carlos Cetina, ccetina@usj.es, SVIT Research Group, Universidad San Jorge, Villanueva de Gállego (Zaragoza), Spain.

Manuscript submitted to ACM

software engineers who work on maintaining the software products. However, software engineers are confronted with the following question once they decide to collaborate: Does some criterion work better for selecting the team of software engineers who collaborate in common maintenance tasks such as feature location?

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 [13, 36, 42, 55].

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. Our 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 [58]. 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 [7].

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 account the profiles for the selection of the participants' queries. Once the queries are selected, collaboration is achieved by applying automatic query reformulation [23, 37], and the search for the relevant model fragment relies on an Evolutionary Algorithm that is guided by similitude to the resulting query [44, 46]. This Evolutionary Algorithm establishes the model fragment that implements a specific requirement (TLR), identifies the model fragment that causes a particular error (BL), or identifies the model fragment that is associated with a specific 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 [57]: recall, precision, and F-measure. Moreover, we provide evidences of the significance 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 confidence 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 confidence level, both the confidence level and an estimated coverage of the feature should be requested. Even though engineers report low confidence 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.

[1] www.caf.net/en

In addition, our results confirm the Defect Principle [70]. In BL, engineers who performed the latest modifications 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 influence 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-Specific 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. Fig. 1 depicts a basic example of a model using an equipment-focused, simplified 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 figure (the train's air conditioning system), and the PA (public address system) and CCTV (television system) on the right.

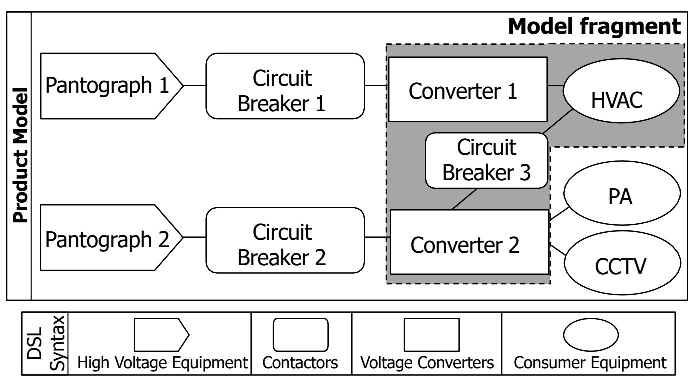

Fig. 1. Example of a product model and a model fragment

Fig. 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 first 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-influenced 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 Fig. 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 specification 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 [17, 18, 46] suggest the use of Search-based Software Engineering [24] 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 [44]. 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 [46] is to obtain the most relevant model fragment (i.e., set of model elements) for a specific TLR, BL, or FL query. To leverage collaboration, the query is obtained by automatically reformulating different software engineers' search queries [44]. 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 influence of the selection criteria on the quality of the retrieved model fragment. The results will serve to recommend the profile that software engineers should have in order to be involved in the collaboration of TLR, BL, and FL tasks. Fig. 2 presents an overview of our work. The left part of the figure 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 profile in terms of a self-rated confidence level (Likert scale). We request a self-rated confidence level to identify relevant search queries. In addition, for each search query, as profile information the engineer provides the number of days from his/her last modification 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

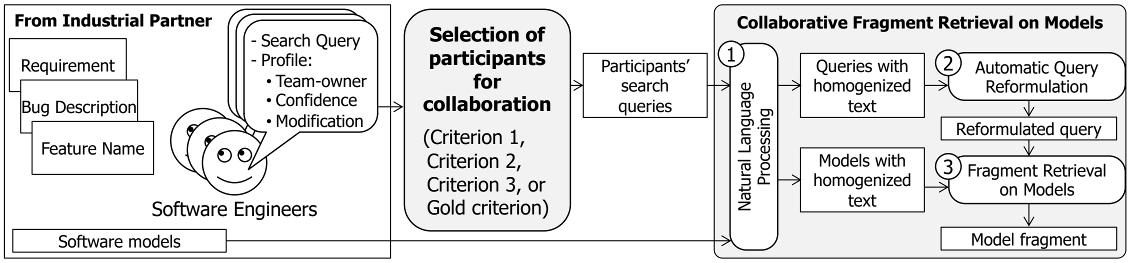

Fig. 2. Overview of our work

Localization Principle) states that the most recent modifications to a project are the most relevant for certain Information Retrieval purposes [25, 62, 70].

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 specification 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 Fig. 2). After the selection of participants for collaboration, their search queries are used as input to perform collaborative fragment retrieval on models as Fig. 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. Specifically, 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 confident engineers. It selects software engineers who have the highest scores in self-rated confidence for the given requirement, bug description, or feature name. The software engineer provides the self-rated confidence level from 7 (highest self-rated confidence) to 1 (lowest self-rated confidence).

Criterion 3: The latest modifications. It selects software engineers who have performed the most recent modifications 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 modified a long time ago. To normalize 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 [70].

Although the next criterion (the 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 confident owners who performed the latest modification. This criterion selects software engineers who have the highest scores in self-rated confidence, perform the latest modifications 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 configuration parameter (*N*) ranging from 2 to the maximum number of engineers who are considered for collaboration.

Other research works [6, 22, 29–32, 38–40, 50, 53, 54] have also reported that these criteria are being used in industry: available owners [22, 53, 54], confidence [6, 29, 30, 32, 40], and the latest modifications [38, 39]. 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, a frequent and beneficial practice [28], through state-of-the-art NLP techniques. Firstly, the text is tokenized (i.e., divided into words) using mainly a white space tokenizer. For some of the sources (i.e., those that use CamelCase naming), more complex tokenizers need to be applied. Secondly, a Parts-of-Speech (POS) tagging technique is used to analyze the words grammatically, inferring the role of each word in the text. Thanks to POS tagging, words are tagged in categories, and those that do not provide semantic information (such as prepositions) can be removed. Then, stemming techniques unify the language by reducing the words to their roots (for instance, plurals are turned into singulars, such as *circuits* to *circuit*), thus enabling the grouping of different words that refer to similar concepts. Finally, the Domain Term Extraction and Stopwords Removal techniques are applied to automatically filter terms in or out of the queries. Towards this last step, 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 whatsoever.

As an example, the following feature description of the industrial partner *The breaker changes to another converter in case of failure in the HVAC converter"* 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 [9], 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 Natural Language (NL) because in software words do not share the same relationships [64]; 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, the most commonly used method for query reformulation [23, 37, 44]. 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: $Rocchio = \sum_{d \in R} TfIdf(t, d)$, 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 first 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 specificity 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 first 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 five 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 [44, 46] 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 [18]. 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 [44, 46].

- The **crossover operation** combines the genetic material from two parent model fragments to create a new individual. The operation looks for the model fragment from Parent 1 in the complete model from Parent 2. If the comparison does not find the model fragment in Parent 2, the crossover returns Parent 1 (the model fragment) unchanged. However, if the model fragment is found in the complete model from Parent 2, the process creates a new individual with the model fragment from Parent 1, albeit referencing the complete model from Parent 2. While both model fragments (the one from Parent 1 and the one from the new individual) will be the same, since each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations. This operation enables the expansion of the search space to a different model.

- The **mutation operator** imitates the random mutations that occur in nature when new individuals are born by adding or removing elements from the model fragment. If the operator is applied to add an element, one element directly related to the elements of the model fragment is added to the model fragment. If the operator is applied to remove an element, a single 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 of the approach assesses the relevance of each of the candidate model fragments by ranking them according to a fitness function. The objective of the fitness function is the similitude between the model fragment and the reformulated query. To do this, we apply methods based on Information Retrieval (IR) techniques. Specifically, the relationships between the model fragments in the population and the reformulated query are analyzed through Latent Semantic Indexing (LSI) [27, 34].

LSI constructs vector representations of a query and a corpus of text documents, 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 three matrices using a matrix factorization technique called Singular Value Decomposition (SVD) [34], with one of the matrices containing a set of vectors that represent the latent semantics of the NL texts. This way, one vector 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 their vectors, obtaining values between -1 and 1. The fitness function is as follows: $fitness(d_1) = \cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$. In the fitness function, 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 θ.

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 the solutions as a starting point that they can use for creating manually refined solutions. They may also refine the query to automatically obtain altogether different solutions.

Note that the focus of this work is on how real-world criteria for selecting software engineers for collaboration affect the quality of maintenance tasks. 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 for selecting engineers for collaboration and the baseline in maintenance tasks (TLR, BL, and FL)?*

RQ$_2$: *Is the difference in the quality of the retrieved model fragment between the different criteria and the baseline significant?*

RQ$_3$: *How much is the quality of the retrieved model fragment influenced using each criterion?*

6.2 Planning and execution

Fig. 3 shows an overview of the methodology that was planned to answer the research questions, which is described 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 specified using a DSL for Train Control and Management, which conforms to MOF (the OMG metalanguage for defining modeling languages that is widely used in the modelling community). The industrial supplier uses the product models to generate the firmware 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. Specifically, 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.

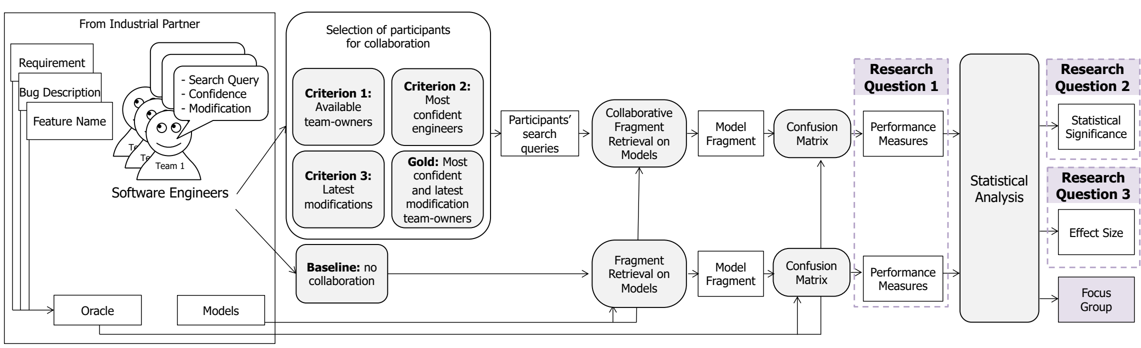

Fig. 3. Methodology to answer each research question

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 confidence and latest modification) 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 confidence level. We chose those engineers with the highest confidence 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 [5] (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 table that describes the performance of a classification model on a set of test data (the best solutions) when the real values are known (from the oracle). In our case, each solution obtained is a model fragment that contains a subset of the model elements that are part of the original complete product model. Therefore, the granularity for the performance of the classification model is at the level of the model elements. Hence, the presence or absence of model elements is considered as a classification for the confusion matrix, which makes distinctions between the predicted values and the real values by classifying them into four categories: (1) True Positive (TP) values, predicted as true by the solution and also true in the oracle real scenario; (2) False Positive (FP) values, predicted as true by the solution but false in the oracle real scenario; (3) True Negative (TN) values, predicted as false by the solution and also false in the oracle real scenario; and (4) False Negative (FN) values, predicted as false by the solution but true in the oracle real scenario.

From the comparison, we obtain a report that includes the following performance measures, widely accepted in the software engineering research community [57]: recall = $\frac{TP}{TP+FN}$ (measuring the percentage of elements of the oracle that are correctly retrieved), precision = $\frac{TP}{TP+FP}$ (measuring the percentage of elements from the solution that are correct according to the oracle), and F-measure = $2 * \frac{Precision*Recall}{Precision+Recall}$ (harmonic mean of precision and recall). Recall and precision values can range from 0% to 100%, with values of 100% precision and 100% recall implying that solution and oracle are identical.

6.2.2 *Answering* RQ_2.

Results should be properly compared in order to establish whether the difference in the quality of the solution between the different criteria and the baseline is significant in TLR, BL, and FL. To that extent, the data resulting from the empirical analysis was analyzed through statistical methods, following the guidelines presented by Arcuri et. al. in [4]. The aim of such an analysis is to provide formal and quantitative evidence (that is, statistical significance) that the criteria and the baseline do in fact have an impact on the comparison metrics (or in other words, that the differences were not obtained by mere chance).

The statistical tests provide *p-value*, which obtains values between 0 and 1 representing probability. The lower the *p-value* of a test, the more likely that we can falsify the null hypothesis H_0 (which states that there is no difference among the criteria and the baseline). The research community considers that a *p-value* under 0.05 is statistically significant towards falsifying the null hypothesis [4].

The test that must be followed depends on the characteristics of the data under study. Our data does not follow a normal distribution, and hence, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data. Among them, the Quade test has proven more powerful when working with real data [20] and has provided better results than the others when the number of algorithms is as low as 4 or 5 algorithms [12],

To determine whether a criterion has a significant impact in the quality of the solution, its outcome should be statistically compared against the outcomes from all of the other 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 the influence of each criterion on the quality of the solution, it is important to assess if a criterion is statistically better than another one, and if so, the magnitude of the improvement. This is achieved through *effect size* measures. We chose the non-parametric Vargha and Delaney's $\hat{A}_{12}$ [21, 65] measure, which measures the probability that running one criterion yields higher values than running another criterion. If the two criteria are equivalent, then the $\hat{A}_{12}$ value will be 0.5. For instance, an $\hat{A}_{12}$ value of 0.7 between Criterion 1 and Criterion 2 would indicate that Criterion 1 obtains better results in 70% of the runs, and an $\hat{A}_{12}$ value of 0.3 would indicate that Criterion 2 obtains better results in 70% of the runs. We recorded an $\hat{A}_{12}$ value for every pair of criteria as well as for every criteria and the baseline in TLR, BL, and FL.

6.3 Implementation details

We implemented the selection of the engineers' queries that are involved in the 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 first 10 term suggestions to expand the base query. We principally chose these values by following the recommendation of the domain literature [9, 44, 46].

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 [1] for the POS-Tagger and the English (Porter2) stemming algorithm [3] for the stemming algorithm (originally created using snowball and then compiled to Java). The LSI was implemented using the Efficient Java Matrix Library (EJML [2]).

The genetic operations are built upon the Watchmaker Framework for Evolutionary Computation [14]. The configuration parameters for the algorithm are as follows: the number of generations (i.e., repetitions of the genetic operations and fitness 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 μ 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 [44, 46].

We are limited by the confidentiality 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 files 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.

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

	Recall ± (σ)		
	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 ± (σ)		
	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 ± (σ)		
	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

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 confident) 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 modifications) obtained the best result in terms of recall, precision, and F-measure (72.31%, 66.34%, and 68.01%, respectively). In FL, the Gold criterion (most confident and latest modification owners) obtained the best result, providing an average value of 91.31% in recall, 92.69% in precision, and 91.83% in F-measure.

7.2 Research Question 2

The p-$Values$ 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 significant 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-$Values$ 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-$Values$ are lower than their corresponding significance threshold value (0.05), indicating that the differences in performance using the criteria are significant. However, some values are greater than the threshold, indicating that the differences between those pair-wise comparisons are not significant.

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

Holm's post hoc p-$Values$	TLR			BL			FL		
	Recall	Precision	F-measure	Recall	Precision	F-measure	Recall	Precision	F-measure
C1 vs C2	$5.5x10^{-11}$	$4.2x10^{-10}$	$7x10^{-15}$	0.024	0.45	0.19	$8.4x10^{-13}$	$3.7x10^{-9}$	$2x10^{-14}$
C1 vs C3	$7.7x10^{-8}$	$5.1x10^{-6}$	$5.2x10^{-10}$	$7.3x10^{-11}$	$2.1x10^{-13}$	$4.1x10^{-14}$	0.67	0.07	0.23
C1 vs Gold	0.62	0.76	0.68	$1x10^{-1}$	$1.2x10^{-1}$	$1.5x10^{-10}$	$1.3x10^{-13}$	$3x10^{-10}$	$2x10^{-14}$
C1 vs Baseline	$1.7x10^{-9}$	$8.4x10^{-8}$	$3.2x10^{-12}$	0.0024	0.068	0.0016	0.32	0.004	0.021
C2 vs C3	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.6x10^{-11}$	$3.5x10^{-13}$	$3.1x10^{-14}$	$8.6x10^{-12}$	$3.9x10^{-11}$	$5.9x10^{-14}$
C2 vs Gold	$3.7x10^{-12}$	$3.6x10^{-16}$	$3.6x10^{-16}$	$3x10^{-10}$	$2.5x10^{-7}$	$1.5x10^{-12}$	0.35	0.53	0.26
C2 vs Baseline	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.18	0.36	0.06	$2.6x10^{-13}$	$1.3x10^{-12}$	$1.5x10^{-14}$
C3 vs Gold	$7.3x10^{-7}$	$4x10^{-7}$	$6.9x10^{-9}$	0.036	0.028	0.0016	$8.6x10^{-12}$	$2.6x10^{-13}$	$2x10^{-14}$
C3 vs Baseline	0.13	0.12	0.034	$7.1x10^{-14}$	$3.1x10^{-14}$	$3.1x10^{-14}$	0.48	0.2	0.15
Gold vs Baseline	$3.3x10^{-10}$	$1x10^{-10}$	$8.5x10^{-15}$	$2.3x10^{-11}$	$8.5x10^{-12}$	$9.4x10^{-14}$	$3.8x10^{-12}$	$7.6x10^{-14}$	$1.3x10^{-13}$

$\hat{A}_{12}$ 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

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

RQ_3 answer. From the results, we can conclude how much the quality of the solution was influenced using each criterion 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 the Gold criterion show a pronounced superiority over Criterion 1, Criterion 3, and the baseline. The largest difference is obtained when comparing Criterion 3 and the Gold criterion (0.0857 for recall, 0.0776 for precision, and 0.0092 for F-measure). Therefore, the 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).

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 [15] 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 first 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 specified 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 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 confidence levels. A high confidence 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 modifications to the models.

A high confidence level and the participation in recent modifications sounds like the perfect profile for collaborating in BL. However, a low confidence level and the participation in recent modifications was also relevant for a significant number of cases. Therefore, we can only state that participating in recent modifications should be specially considered for collaboration in the context of BL. This recommendation is aligned with the finding of the Defect Principle, which states that the most recent modifications of a project are the most relevant for certain Information Retrieval purposes [70].

For FL, the 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 confidence 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 confidence 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 confidence level.

Our analysis of the results reveals that the confidence 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 confidence level. Therefore, we should also ask software engineers about the percentage of coverage that they think their search query may achieve, and, consequently, the confidence 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. Specifically, 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 confident owners who performed the latest modification) 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 counter-intuitive because they thought that those engineers who are not confident (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 confident 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 difficult 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 classification of threats of validity of [56, 68], 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 [57].

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 [44, 46]. 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 [9, 44, 46]. As suggested by Arcuri and Fraser [5], 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 defining 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 confidence, and latest modification). 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 [26, 44, 63], 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 specific 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 [60, 69]. Yue et al. [69] developed a web search system to investigate factors that influence 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 [59]. Most existing research is focused on query expansion by finding terms in relevant documents such as source code and Internet sites [8, 26, 35, 41, 51, 52, 61, 67]. Sirres et al. [61] propose a technique for finding relevant code using free-form query terms from internet sites such as Q&A posts from Stack Overflow. Hill et al. [26] extract possible query expansion terms from the code using word context. Cao et al. [8] propose an automated query reformulation approach for efficient search using query logs provided by Stack Overflow.

In contrast, other approaches propose automatically reformulating the query by removing words to reduce long queries. Chaparro et al. [10] 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. [11] 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 [33] analyze the most promising subsets of terms from the original query to reduce queries. Haiduc et al. [23] 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. [16] 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. [66] propose a code search approach, which incorporates user feedback to refine 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. 4 shows the connections and differences between our previously published works and this work. At the bottom of the figure, it is possible to find our work regarding Search-Based Software Engineering (SBSE) [49]. While [49] 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.

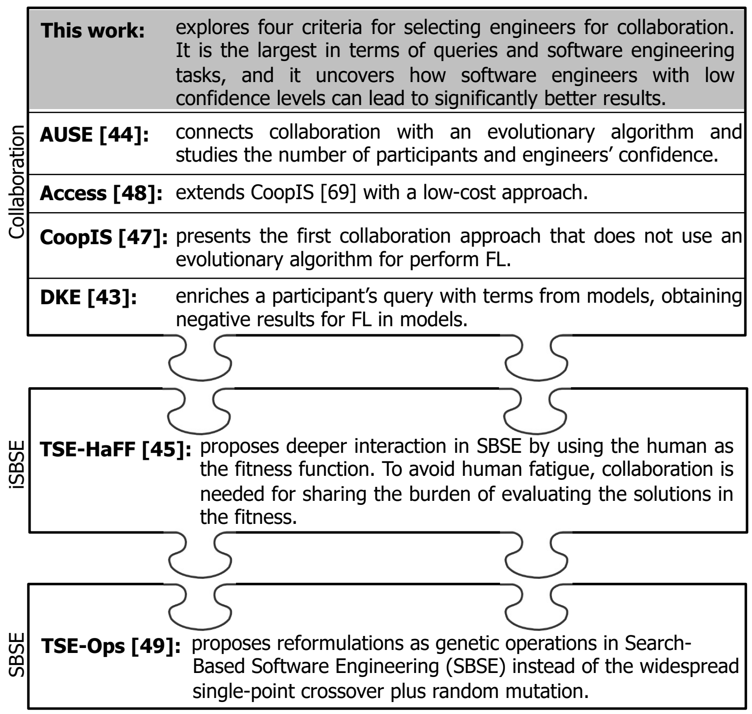

Fig. 4. Comparing this work with our previous works

The middle part of Fig. 4 shows the work in [45], where the human plays the role of the fitness function of the evolutionary algorithm. As one of its outcomes, the work in [45] 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. 4 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 [43], 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 [47] was our first approach where the criterion of the most confident engineer is applied to address collaboration among different engineers (without an evolutionary algorithm to perform the location of features). Through the work presented in [48], the work in [47] 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 [44] 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' confidence produces an improvement in the results.

Table 3. Comparing our works that address collaboration

	Queries	Criteria	Reformulation techniques	Software engineering tasks
This work	2565	C1: Available owners C2: The most confident engineer C3: The latest modification C4: Gold	Rocchio	FL TLR BL
AUSE [44]	817	C2: The most confident engineer	Rocchio	FL
Access [48]	817	C2: The most confident engineer	Rocchio, RSV, Dice, Reduction	FL
CoopIS [47]	817	C2: The most confident engineer	Rocchio	FL
DKE [43]	217	-	Rocchio, RSV, Dice, Reduction	FL

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 how the quality of the results is affected by different real-world criteria for selecting participants for collaboration (Available owners, The most confident engineer, and The latest modification) 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 4 shows). Moreover, the intuition of our industrial partner suggests that a combination of two criteria (The most confident engineer and The latest modification) should be the criterion that obtains the best results. This new criterion, identified as the Gold criterion, has never been used before by our industrial partner nor by our previous research, and is explored for the first time in this paper. Finally, our work uncovers novel recommendations (even some counter-intuitive 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 how collaboration affects 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. In any case, it is not for us to claim that collaboration should be systematically applied to every case. Rather, collaboration becomes necessary when the requirement/bug/feature significantly 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, counter-intuitively 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 VARIATIVA under Grant PID2021-128695OB-I00, and in part by the Gobierno de Aragón (Spain) (Research Group S05_20D).

REFERENCES

[1] 2021. Apache OpenNLP: Toolkit for the processing of natural language text. https://opennlp.apache.org/.

[2] 2021. Efficient Java Matrix Library. http://ejml.org/.

[3] 2021. English (Porter2) stemming algorithm. http://snowball.tartarus.org/algorithms/english/stemmer.html.

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

[5] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18, 3 (2013), 594–623. https://doi.org/10.1007/s10664-013-9249-9

[6] Matthew Bass, James D. Herbsleb, and Christian Lescher. 2007. 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*. 33–39. https://doi.org/10.1109/ICGSE.2007.16

[7] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. *Model-Driven Software Engineering in Practice* (1st ed.). Morgan & Claypool Publishers.

[8] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. 2021. Automated Query Reformulation for Efficient Search Based on Query Logs From Stack Overflow. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1273–1285. https://doi.org/10.1109/ICSE43902.2021.00116

[9] Claudio Carpineto and Giovanni Romano. 2012. A Survey of Automatic Query Expansion in Information Retrieval. *ACM Comput. Surv.* 44, 1, Article 1 (Jan. 2012), 50 pages.

[10] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2018. 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. 376–387. https://doi.org/10.1109/ICSME.2017.100

[11] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2019. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. *Empirical Software Engineering* 24, 5 (2019), 2947–3007. https://doi.org/10.1007/s10664-018-9672-z

[12] William Jay Conover. 1999. *Practical nonparametric statistics* (3. ed ed.). Wiley, New York, NY [u.a.].

[13] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2011. Feature Location in Source Code: A Taxonomy and Survey. In *Journal of Software Maintenance and Evolution: Research and Practice.*

[14] Daniel Dyer. 2016. The Watchmaker Framework for Evolutionary Computation (evolutionary/genetic algorithms for Java). http://watchmaker.uncommons.org/. [Online; accessed 7-April-2016].

[15] Jorge Echeverría, Francisca Pérez, José Ignacio Panach, Carlos Cetina, and Oscar Pastor. 2017. 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.*

[16] Juan Manuel Florez, Oscar Chaparro, Christoph Treude, and Andrian Marcus. 2021. Combining Query Reduction and Expansion for Text-Retrieval-Based Bug Localization. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 166–176. https://doi.org/10.1109/SANER50967.2021.00024

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

(Saint-malo, France) *(MODELS '16)*. ACM, New York, NY, USA, 272–282. https://doi.org/10.1145/2976767.2976789

[18] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2017. Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering. *IEEE Transactions on Evolutionary Computation* PP, 99 (2017), 1–1. https://doi.org/10.1109/TEVC.2017.2751100

[19] Mirco Franzago, Davide Di Ruscio, Ivano Malavolta, and Henry Muccini. 2017. Collaborative Model-Driven Software Engineering: a Classification Framework and a Research Map. *IEEE Transactions on Software Engineering* PP, 99 (2017), 1–1. https://doi.org/10.1109/TSE.2017.2755039

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

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

[22] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2011. "Not My Bug!" and Other Reasons for Software Bug Report Reassignments. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work* (Hangzhou, China) *(CSCW '11)*. ACM, 395–404. https://doi.org/10.1145/1958824.1958887

[23] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic Query Reformulations for Text Retrieval in Software Engineering. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. 842–851.

[24] Mark Harman and Bryan F. Jones. 2001. Search-based software engineering. *Information & Software Technology* 43, 14 (2001), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

[25] Ahmed E. Hassan and Richard C. Holt. 2005. The Top Ten List: Dynamic Fault Prediction. In *21st IEEE International Conference on Software Maintenance*.

[26] Emily Hill, Lori Pollock, and K. Vijay-Shanker. 2009. Automatically Capturing Source Code Context of NL-queries for Software Maintenance and Reuse. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. 232–242. https://doi.org/10.1109/ICSE.2009.5070524

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

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

[29] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. 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* (Amsterdam, The Netherlands) *(ESEC/FSE '09)*. ACM, New York, NY, USA, 111–120. https://doi.org/10.1145/1595696.1595715

[30] Huzefa Kagdi and Denys Poshyvanyk. 2009. Who can help me with this change request?. In *2009 IEEE 17th International Conference on Program Comprehension (ICPC 2009)*. https://doi.org/10.1109/ICPC.2009.5090056

[31] Muhammad Rezaul Karim, Günther Ruhe, Md. Mainur Rahman, Vahid Garousi, and Thomas Zimmermann. 2016. An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *Journal of Software: Evolution and Process* 28, 12 (2016), 1025–1060. https://doi.org/10.1002/smr.1777

[32] Katja Kevic, Sebastian C. Müller, Thomas Fritz, and Harald C. Gall. 2013. Collaborative bug triaging using textual similarities and change set analysis. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 17–24. https://doi.org/10.1109/CHASE.2013.6614727

[33] Giridhar Kumaran and Vitor R. Carvalho. 2009. Reducing Long Queries Using Query Quality Predictors. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Boston, MA, USA) *(SIGIR '09)*. ACM, New York, NY, USA, 564–571. https://doi.org/10.1145/1571941.1572038

[34] Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An Introduction to Latent Semantic Analysis. *Discourse processes* 25 (1998).

[35] Zhixing Li, Tao Wang, Yang Zhang, Yun Zhan, and Gang Yin. 2016. Query Reformulation by Leveraging Crowd Wisdom for Scenario-based Software Search. In *Proceedings of Internetware* (Beijing, China). 36–44.

[36] Anas Mahmoud, Nan Niu, and Songhua Xu. 2012. A Semantic Relatedness Approach for Traceability Link Recovery. In *IEEE 20th International Conference on Program Comprehension*.

[37] Lee Martie, Thomas D. LaToza, and André van der Hoek. 2015. CodeExchange: Supporting Reformulation of Internet-Scale Code Queries in Context. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE*. 24–35. https://doi.org/10.1109/ASE.2015.51

[38] David W. McDonald and Mark S. Ackerman. 2000. Expertise recommender: a flexible recommendation system and architecture.. In *CSCW*. ACM, 231–240.

[39] Audris Mockus and James D. Herbsleb. 2002. 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*. 503–512. https://doi.org/10.1145/581339.581401

[40] Kumiyo Nakakoji. 2006. Supporting Software Development as Collective Creative Knowledge Work. In *2nd International Workshop on Supporting Knowledge Collaboration in Software Development*.

[41] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Transactions on Services Computing* 9, 5 (2016), 771–783. https://doi.org/10.1109/TSC.2016.2560165

[42] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. 2010. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *IEEE 18th International Conference on Program Comprehension*.

[43] Francisca Pérez, Jaime Font, Lorena Arcega, and Carlos Cetina. 2018. Automatic Query Reformulations for Feature Location in a Model-based Family of Software Products. *Data & Knowledge Engineering* (2018). https://doi.org/10.1016/j.datak.2018.06.001

[44] Francisca Pérez, Jaime Font, Lorena Arcega, and Carlos Cetina. 2019. Collaborative feature location in models through automatic query expansion. *Automated Software Engineering* 26, 1 (2019), 161–202. https://doi.org/10.1007/s10515-019-00251-9

[45] Francisca Pérez, Jaime Font, Lorena Arcega, and Carlos Cetina. 2021. Empowering the Human as the Fitness Function in Search-Based Model-Driven Engineering. *IEEE Transactions on Software Engineering* 01 (oct 2021), 1–1. https://doi.org/10.1109/TSE.2021.3121253

[46] Francisca Pérez, Raúl Lapeña, Jaime Font, and Carlos Cetina. 2018. Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study. *Information & Software Technology* 103 (2018), 188–201. https://doi.org/10.1016/j.infsof.2018.06.017

[47] Francisca Pérez, Ana Cristina Marcén, Raúl Lapeña, and Carlos Cetina. 2017. Introducing Collaboration for Locating Features in Models: Approach and Industrial Evaluation. In *Proceedings of the 25th International Conference on Cooperative Information Systems, CoopIS*. 114–131. https://doi.org/10.1007/978-3-319-69462-7_9

[48] Francisca Pérez, Ana Cristina Marcén, Raúl Lapeña, and Carlos Cetina. 2020. Evaluating Low-Cost in Internal Crowdsourcing for Software Engineering: The Case of Feature Location in an Industrial Environment. *IEEE Access* 8 (2020), 65745–65757. https://doi.org/10.1109/ACCESS.2020.2985915

[49] Francisca Pérez, Tewfik Ziadi, and Carlos Cetina. 2020. Utilizing Automatic Query Reformulations as Genetic Operations to Improve Feature Location in Software Models. *IEEE Transactions on Software Engineering* 01 (jun 2020), 1–1. https://doi.org/10.1109/TSE.2020.3000520

[50] Md. Mainur Rahman, Muhammad Rezaul Karim, Guenther Ruhe, Vahid Garousi, and Thomas Zimmermann. 2015. 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* (Dearborn, Michigan, USA).

[51] Mohammad Masudur Rahman and Chanchal Roy. 2018. 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)*.

[52] Mohammad Masudur Rahman and Chanchal K. Roy. 2016. QUICKAR: Automatic Query Reformulation for Concept Location Using Crowdsourced Knowledge. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore). 220–225.

[53] Md. Mainur Rahman, Günther Ruhe, and Thomas Zimmermann. 2009. 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*. 439–442.

[54] Md. Mainur Rahman, S. M. Sohan, Frank Maurer, and Günther Ruhe. 2010. 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*.

[55] Julia Rubin and Marsha Chechik. 2013. A survey of feature location techniques. In *Domain Engineering*. Springer, 29–58.

[56] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131–164.

[57] Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

[58] Bran Selic. 2003. The Pragmatics of Model-Driven Development. *IEEE Softw.* 20, 5 (Sept. 2003), 19–25. https://doi.org/10.1109/MS.2003.1231146

[59] Chirag Shah. 2010. Collaborative Information Seeking: A Literature Review. *Exploring The Digital Frontier Advances In Librarianship* 32 (2010).

[60] Chirag Shah and Roberto González-Ibáñez. 2011. 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* (Beijing, China) *(SIGIR '11)*. ACM, New York, NY, USA, 913–922. https://doi.org/10.1145/2009916.2010038

[61] Raphael Sirres, Tegawendé F. Bissyandé, Dongsun Kim, David Lo, Jacques Klein, Kisub Kim, and Yves Le Traon. 2018. Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering* 23, 5 (2018), 2622–2654. https://doi.org/10.1007/s10664-017-9544-y

[62] Bunyamin Sisman and Avinash C. Kak. 2012. Incorporating Version Histories in Information Retrieval Based Bug Localization. In *9th IEEE Working Conference on Mining Software Repositories*.

[63] Bunyamin Sisman and Avinash C. Kak. 2013. Assisting code search with automatic query reformulation for bug localization. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR*. 309–318. https://doi.org/10.1109/MSR.2013.6624044

[64] Giriprasad Sridhara, Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. 2008. Identifying Word Relations in Software: A Comparative Study of Semantic Similarity Tools.. In *ICPC*, René L. Krikhaar, Ralf Lämmel, and Chris Verhoef (Eds.). IEEE Computer Society, 123–132.

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

[66] Shaowei Wang, David Lo, and Lingxiao Jiang. 2014. Active Code Search: Incorporating User Feedback to Improve Code Search Relevance. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. 677–682. https://doi.org/10.1145/2642937.2642947

[67] Wentao Wang, Arushi Gupta, Nan Niu, Li Da Xu, Jing-Ru C. Cheng, and Zhendong Niu. 2018. Automatically Tracing Dependability Requirements via Term-Based Relevance Feedback. *IEEE Trans. Industrial Informatics* 14, 1 (2018), 342–349. https://doi.org/10.1109/TII.2016.2637166

[68] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*.

[69] Zhen Yue, Shuguang Han, Daqing He, and Jiepu Jiang. 2014. Influences on Query Reformulation in Collaborative Web Search. *Computer* 47, 3 (Mar. 2014), 46–53. https://doi.org/10.1109/MC.2014.62

[70] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. 2004. Mining Version Histories to Guide Software Changes. In *Proceedings of the 26th International Conference on Software Engineering*.