Automatic Query Reformulations for Feature Location in a Model-based Family of Software Products

Francisca Pérez[a,*], Jaime Font[a,b], Lorena Arcega[a,b], Carlos Cetina[a]

[a] *SVIT Research Group, Universidad San Jorge.*
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Villanueva de Gállego (Zaragoza), Spain
[b] *Department of Informatics, University of Oslo.*
Postboks 1080 Blindern, 0316 Oslo, Norway

Abstract

No maintenance activity can be completed without Feature Location (FL), which is finding the set of software artifacts that realize a particular functionally. Despite the importance of FL, the vast majority of work has been focused on retrieving code, whereas other software artifacts such as the models have been neglected. Furthermore, locating a piece of information from a query in a large repository is a challenging task as it requires knowledge of the vocabulary used in the software artifacts. This can be alleviated by automatically reformulating the query (adding or removing terms). In this paper, we test four existing query reformulation techniques, which perform the best for FL in code but have never been used for FL in models. Specifically, we test these techniques in two industrial domains: a model-based family of firmwares for induction hobs, and a model-based family of PLC software to control trains. We compare the results provided by our FL approach using the query and the reformulated queries by means of statistical analysis. Our results show that reformulated queries do not improve the performance in models, which could lead towards a new direction in the creation or reconsideration of these techniques to be applied in models.

Keywords: Conceptual modeling, Information retrieval, Feature Location, Query Reformulation, Software Maintenance and Evolution, Families of Software Products

1. Introduction

Feature location (FL) is known as the process of finding the set of software artifacts that realize a particular functionality of software system. No maintenance activity can be completed without locating in the first place the software artifact (e.g., code) that is relevant to the specific functionality [1]. Since FL is one of the main activities performed during software evolution [1] and up to an 80% of a system's lifetime is spent on the maintenance

[*] Corresponding author. Tel.: +34 976060100

Email addresses: mfperez@usj.es (Francisca Pérez), jfont@usj.es (Jaime Font), larcega@usj.es (Lorena Arcega), ccetina@usj.es (Carlos Cetina)

and evolution of the system [2], there is a great demand for FL approaches that can help developers to find relevant software artifacts in a family of software products.

Many of FL approaches use of Information Retrieval (IR) techniques [1, 3] such as Latent Semantic Indexing (LSI) [4], Latent Dirichlet Allocation (LDA) [5], and Vector Space Model [6] and involve the formulation of a query in natural language (e.g., by the developer). These techniques are statistical methods used to find a feature's relevant software artifact by analyzing and retrieving words that are similar to a query provided by a user. For example, during FL, a developer formulates a query which describes the feature to be located in the code. The query is then run by the IR technique and a list of ranked software artifacts (e.g., classes or methods) is retrieved.

The performance of the retrieval depends greatly on the textual query and its relationship to the text contained in the software artifacts [7]. Hence, this relationship requires knowledge of the vocabulary of the software artifacts to be searched. This knowledge can be difficult to acquire in industrial environments that accumulate a vast amount of software over the years, which often emerges ad hoc using software reuse techniques such as duplication (the "clone-and-own" approach) instead of formalizing the variability among the family of software products. Moreover, in these industrial environments, software maintenance tasks are performed by people who have not participated during the development, so the vocabulary that the people use on the textual query to locate features during maintenance tasks can differ from the vocabulary that was used during the development. Therefore, these differences between the query and the text contained in the software artifacts make the performance of the retrieval worse.

To overcome these differences between the query and the text contained in the software artifacts, other FL approaches [7, 8, 9] refine the query using automatic reformulation techniques: expansion or reduction. A short query which obtains not relevant results will likely need an expansion strategy (i.e., adding terms) to improve its performance, whereas a verbose query may need a reduction strategy (i.e., removing terms) since the performance uses to be deteriorated handling long queries [10].

To date the vast majority of work in FL has been focused on improving the performance of the retrieval using automatic query reformulation strategies in code (i.e., by better performance we mean retrieving the relevant software artifacts closer to the top of the list of results). Nevertheless, other software artifacts such as the models have been neglected even though models are the cornerstone in Model-Driven Development approaches to generate code.

To cope with this lack, we evaluate whether automatic query reformulation strategies could improve the results of FL in models. Therefore, the contribution of this paper is twofold.

1. We test four automatic query reformulation techniques (Query reduction, Rocchio query expansion, RSV query expansion and Dice query expansion), which perform best in that field [7] and have never been used to locate features in models. Specifically, we test these techniques in two industrial domains: the model-based product family of the BSH group (www.bsh-group.com) and the model-based product family of CAF

(www.caf.net/en).
The BSH group is one of the largest manufacturers of home appliances in Europe. Its induction division has been producing Induction Hobs (sold under the brands of Bosch and Siemens) for the last 15 years. CAF produces a family of PLC software to control the trains that they have been developing over more than 25 years.

2. We compare the results provided by our FL approach using the query as it is (baseline) with the results provided by the four reformulation techniques.
The results of this paper suggest that current automatic query reformulation techniques should be reconsidered to be applied in models since we found that using the query as it is leads better results in models than including the query expansion/reduction reformulations. We hope that these results help FL users when they work with models to loss the inertia of applying query reformulation techniques as they would do to locate features in code. Moreover, these results would contribute towards a new direction in the creation of new query reformulation techniques or the modification of the existing ones to improve the location of features in models.

The rest of the paper is structured as follows: Section 2 provides the required background on the automatic query reformulation techniques being compared. Section 3 presents the approach to perform feature location in models. Next, Section 4 presents the evaluation performed, and Section 5 shows the results. Section 6 discusses the results. Section 7 describes the threats to validity. Section 8 reviews the related work. Finally, Section 9 concludes the paper.

2. Query reformulation techniques

Researchers in the field of FL have proposed a large variety of approaches for automatic query reformulations for an initial query. These approaches belong to one of the following two categories [11]: query expansion approaches and query reduction approaches.

Next, we introduce briefly these categories with emphasis on the automatic reformulation strategies that we use for FL in models.

2.1. Query Reduction

Longer queries are typically used to express more sophisticated information needs. Nevertheless, longer queries use to include both important information and noise, i.e., terms that serve more to confuse the search engine that support it in its task. Since the performance of most commercial and academic search engines deteriorates while handling longer queries [10], the aim of query reduction is to reduce long queries to shorter.

A conservative automatic query reduction approach that has been previously used in software engineering [7, 12] consists of eliminating non-discriminating terms, which are those terms that appear in more than 25% of the documents in the corpus.

2.2. Query Expansion

Queries require including terms that fit with the vocabulary of the software artifacts to be searched. Nevertheless, the most critical language issue for performance is that the users often do not use the same words [13].

To deal with this issue, several query expansion techniques have been proposed [13]. Since not all of these techniques can be applied in our work (i.e., model based corpus), we selected three existing techniques using the following criteria: 1) we did not consider techniques that relied on sources of information external to the corpus, like the web, or ontologies; 2) we did not selected techniques that are designed to work for natural language documents as they rely on word relationships that exist in natural language but words do not share the same relationships in software according to previous studies [14]; and 3) we did not consider non-practical techniques that were based on algorithms with high computational complexity to produce reformulations for a query since our goal is to support users' daily FL tasks.

At this point, it is important to note that all three selected strategies consider the top K documents from the list of results as relevant documents to the query. After, different techniques are used to order the terms in these K documents and select the top N terms to expand the query.

The first technique is based on Rocchio's method [15], which is perhaps the most commonly used method for query reformulation [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:

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

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 it 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 indicates the specificity of that term for a document containing it.

The second technique orders the terms in the top K documents using the Robertson Selection Value (RSV). RSV also uses *TfIdf* as part of its equation as the first technique does, but it also considers the probability of a term occurring in a relevant document in order to determine its importance. RSV is given by the following equation:

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

where C denotes the collection of documents in the corpus, 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. Thus, $p(t|R)$ is the number of times that the term t appears in the top K documents in the list of results (R), and $p(t|C)$ is the number of times that t appears in the whole document collection (C).

Note that the first and second techniques do not rely on similarities with the terms in the query. The idea is to use the first K documents retrieved in response to the query to expand it with the most representative terms. In contrast, the third technique orders the terms in the top K relevant documents based on their *Dice similarity* with each term of the query. *Dice similarity* is that two terms are related if they appear in the same documents in the corpus (a common assumption in all FL engines). *Dice similarity* is given by the following equation:

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

where u is a term from the query, v is a term from the top K documents, and df denotes the number of documents in the corpus containing u, v, or both u and v, respectively.

3. Our approach to locate features in a model-based product family

The objective of our approach is to obtain the model fragment in a product family that belongs to a feature description provided as input. The upper part of Figure 1 shows a simplified subset of the Induction Hobs Domain Specific Language (IHDSL) used by our industrial partner BSH to specify induction hobs. We use a simplified subset as a running example through the rest of the paper in order to gain legibility (since the IHDSL is composed of 46 meta-classes, 74 references among them and more than 180 properties) and due to intellectual property rights concerns.

The bottom left part of Figure 1 depicts an example of a product model specified with the IHDSL. The product model contains four inverters used to power two different inductors. The upper inductor is powered by a single inverter while the lower inductor is powered by the combination of three different inverters. Power managers act as hubs to perform the connection between the inverters and the inductors.

The bottom right part of Figure 1 shows an example of a model fragment of the product model (represented with a dashed square). The model fragment includes the three inverters in charge of powering the lower inductor along with the three provider channels and the power manager used to aggregate and manage the power provided by those inverters. This model fragment is the realization of a feature called “triple inverter”.

In order to provide the model fragment from a given product family that realizes a target feature description, our approach relies on an evolutionary algorithm that iterates a population of model fragments and evolves them using genetic operations. The evolutionary algorithm is guided by a fitness operation that takes into account the feature description. As output, the approach provides a rank of model fragments that might be realizing the feature.

In addition, our approach automatically reformulates the target feature description using one of the techniques presented in the previous section. Since locating a model fragment in a product family is a challenging task as it requires knowledge of the vocabulary of the

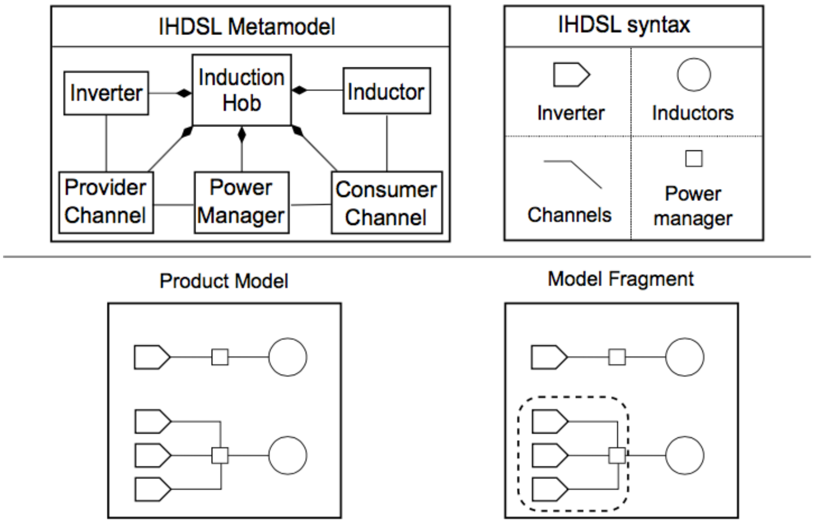


Figure 1: Example of product model and model fragment

software artifacts to be searched [7], the automatic query reformulation could lead to improve the results of the approach by bringing the relevant model fragments closer to the top of results rather than using the feature description as it was provided. The top of Figure 2 shows an example of input to our approach.

- A set of **product models** that contains the target feature. The engineer selects a subset of relevant product models from the entire family of products that include the feature to be located.

- A **feature description** of the target feature using natural language. Typically, these descriptions can come from textual documentation of the products, bug reports, or oral descriptions from the engineers. As occurs in other works [16, 17], results depend greatly on the terms that are included in the feature description. Therefore, the feature description may include some domain-specific terms that are similar to those used when specifying the target feature in the product model using the modeling language. The knowledge of the engineers about the domain and the product models will be useful in selecting the textual description from the sources available.

  The feature description is used as query to reformulate it. The **reformulated query** is obtained by applying one of the techniques presented in the previous section (Query reduction, Rocchio query expansion, RSV query expansion, or Dice query expansion) in order to add or remove terms to the initial query.

The center of Figure 2 shows a simplified representation of the main steps of our approach.

1. The **initialize population** step calculates an initial population of model fragments from the input set of product models. This initial population of model fragments is randomly extracted from the product models.

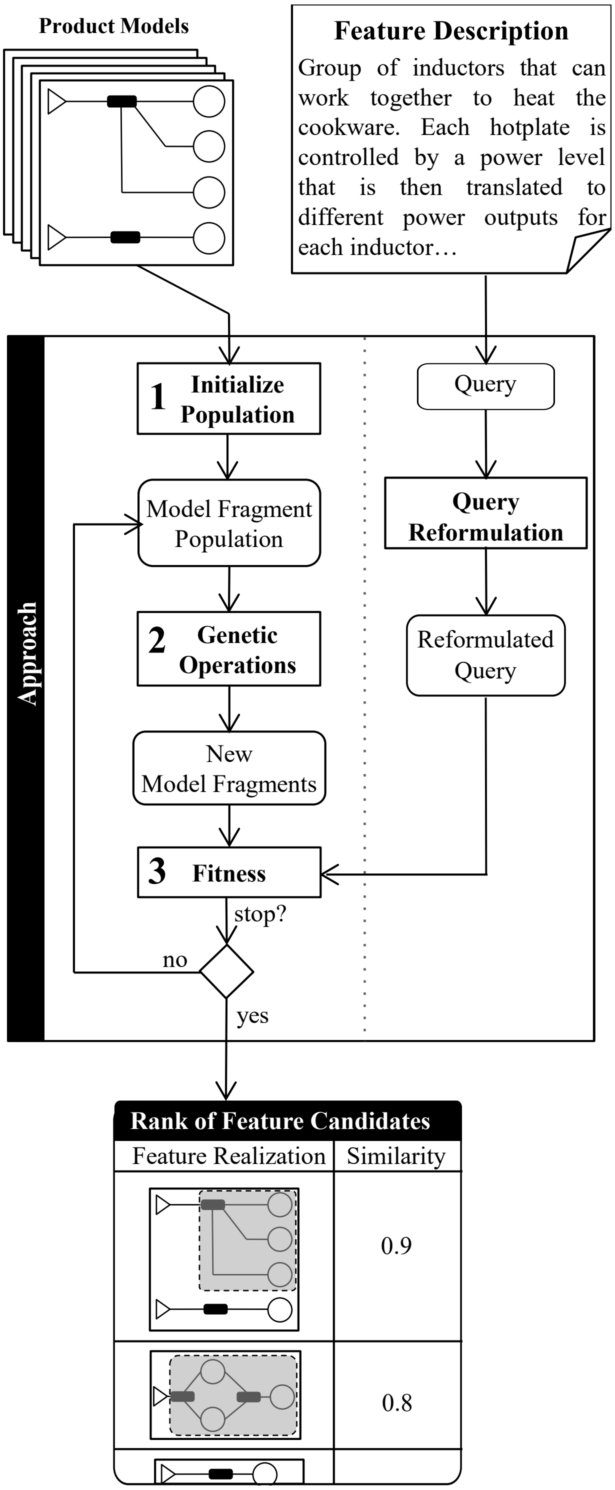

Product Models
Feature Description
Group of inductors that can work together to heat the cookware. Each hotplate is controlled by a power level that is then translated to different power outputs for each inductor…
Approach
1 Initialize Population
Model Fragment Population
2 Genetic Operations
New Model Fragments
3 Fitness
stop?
no
yes
Query
Query Reformulation
Reformulated Query
Rank of Feature Candidates
Feature Realization
Similarity
0.9
0.8

Figure 2: Overview of our approach

2. The **genetic operations** generate the new generation of model fragments. First, a selection operation selects the model fragments that will be used as parents of the new model fragments. The fitness values are used to ensure that the best model fragments are chosen as parents. Then, a crossover operation mixes the model elements of the two parents into a new model fragment. Finally, a mutation operation introduces variations in the new model fragment (by adding or removing model elements) in hopes that the new model fragment achieves better fitness values than its parents.
3. The **fitness** step assigns values that assess how good each model fragment is for the reformulated query. The more terms shared between the reformulated query and the properties of the model fragment, the higher this fitness value.

The bottom of Figure 2 shows the output of our approach, which is a rank of model fragments that realize the target feature. The ranking is ordered according to the similarity to the reformulated query.

As suggested by Davis [18], we decided to use an encoding natural for our problem and then devise genetic operations for that specific encoding. Since the goal of feature location in models is to locate the relevant model fragment for the feature, we encode our individuals as model fragments, and the genetic operations produce model fragment offsprings. The following subsections describe the representation used to encode model fragments, the genetic operations of our approach to generate new model fragments, and how the fitness step determines the suitability of each model fragment in terms of similarity to the reformulated query.

3.1. Model fragment representation

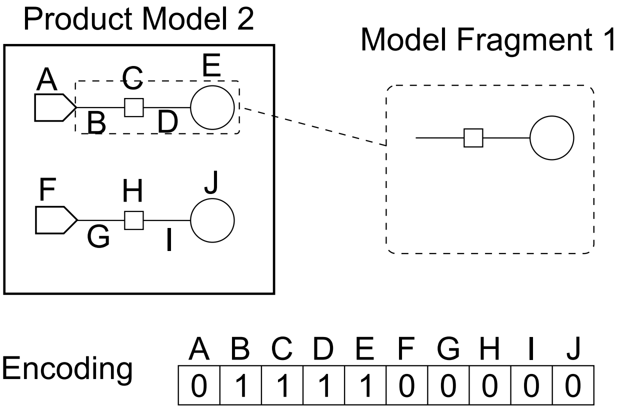
Since our approach provides as a solution a set of model fragments that realize the feature, it is necessary to formalize the model fragments and represent them in order to both generate new model fragments using the genetic operations, and to extract the terms that belong to each model fragment in the fitness step, which assesses how good is each model fragment for the query.

On the one hand, we use the Common Variability Language (CVL) [19] to formalize the model fragments used by the approach given its capabilities to formalize the feature realizations in terms of model fragments. CVL defines variants of a base model (conforming to MOF, the OMG metalanguage for defining modeling languages) by replacing variable parts of the base model (the features) by alternative model replacements (the feature realizations) found in a library.

On the other hand, we encode each model fragment since evolutionary algorithms traditionally encode each possible solution of the problem as a string of binary values. Each position of the string has two possible values: 0 or 1. Each individual of our proposed approach will be a model fragment that is defined in one of the product models. In other words, each individual is a set of model elements that are present in one of the product models.

Figure 3 shows an example of our representation of model fragments. We depict each model element of the product model with a letter. The letters A and F correspond to inverters, the letters B, D, G, and I correspond to channels, and the letters E and J correspond to inductors. Therefore, the string of binary values that represents the model fragment from this product model has the positions that correspond to each letter with a value of 0 or 1. If the model element appears in the model fragment, the value will be 1; if the model element does not appear in the model fragment the value will be 0.

At this point, it is important to note that each model fragment representation depends on the product model that it came from. Throughout the rest of the paper, we will refer to each individual as a model fragment that is part of a product model.



A	B	C	D	E	F	G	H	I	J
0	1	1	1	1	0	0	0	0	0

Figure 3: Representation the model fragment encoding

3.2. Genetic operations

In order to generate a set of model fragments that could realize the feature description, we apply genetic operators adapted to work over model fragments. Specifically, we use two genetic operations: the mutation and the crossover.

Our **crossover operation** mixes two individuals (model fragments) that act as parents to create a new individual. This operator preserves the basic mechanics of the crossover operation and is based on model comparisons.

The upper part of Figure 4 shows an example of the application of the crossover operation on model fragments. First, we select the model fragment from the first parent. Then, we select the product model from the second parent. The model fragment (from first parent) is then compared with the product model (from the second parent). If the comparison finds the model fragment in the product model, the operation creates a new individual with the model fragment taken from the first parent but referencing the product model from the second parent. In the case that the comparison does not find a similar element, the crossover will return the first parent unchanged.

This operation enables the search space to be expanded to a different product model, i.e., both model fragments (the one from the first parent 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.

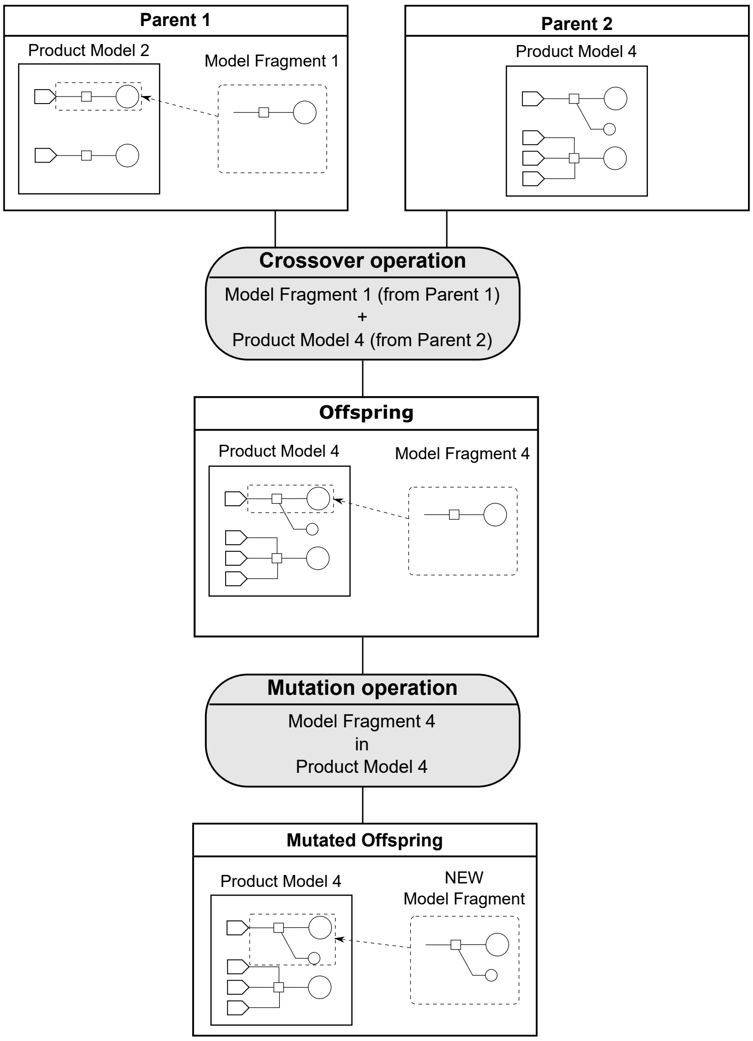


Figure 4: Crossover and Mutation genetic operators over model fragments

Our **mutation operation** creates new model fragments adding or removing elements of a model fragment. The elements of the model fragment to be added/removed are driven by the referenced product model in order to prevent the creation of invalid model fragments. The type of mutation that will occur (either adding or removing elements) is decided randomly and they are the following:

Subtractive Mutation: This kind of mutation randomly removes elements from the model fragment. Specifically, the candidate elements to be removed have to fulfill the constraint of being in the boundaries of the model fragment (i.e., they are connected with a single element). From the candidate elements, it is removed a number of elements that is chosen randomly. Thus, the resulting model fragment is prevented to be invalid because it is a subset of the referenced product model.

Additive Mutation: This kind of mutation randomly adds elements to the model fragment. Specifically, the set of candidate elements have to fulfill the constraint of being part of the referenced product model. From this set, a random number of elements is added to the resulting model fragment. By doing so, the mutated model fragment will be part of the referenced product model, so the creation of invalid model fragments is prevented.

The lower part of Figure 4 shows an example of mutation for model fragments. Specifically, the figure shows the creation of a new model fragment that is part of the referenced Product Model 4 using an additive mutation, which adds an inductor and a channel to the model fragment.

Table 1 shows the settings of our approach such as the size of the population, the number of parents, the probability of the crossover and mutation operation. We have principally chosen values for those settings that are commonly used in the literature [20].

Table 1: Parameter settings of our approach

Parameter	**Description**	**Value**
$Size$	Size of the population	100
StopCondition	budget allocated to run the algorithm (s)	80
μ	Number of Parents	2
λ	Number of offspring from μ parents	1
$p_{crossover}$	Crossover probability	0.75
$p_{mutation}$	Mutation probability, where n is the length of the chromosome being mutated	1/n

As suggested by [21] and confirmed in [22], tuned parameters can outperform default values generally, but they are far from optimal in individual problem instances. Hence, the goal of this paper is not to tune the values to improve the performance of our approach but rather to compare the inclusion of automatic query reformulation techniques using default parameter values. Therefore, we use default parameter values that are commonly used in the literature [20].

3.3. Model fragment fitness

The model fragment fitness consists of assessing the suitability of each candidate model fragment produced as a result of mutations and crossovers, and ranking them according to a fitness function. Each model fragment is assigned with a fitness value that is determined by the similarity to the reformulated query provided as input.

To assess the relevance of each model fragment in relation to the reformulated query, we apply methods based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Analysis (LSA) [23] to analyze the relationships between the reformulated query and the model fragments since most of the efforts show better results when applying LSA [24, 25, 26].

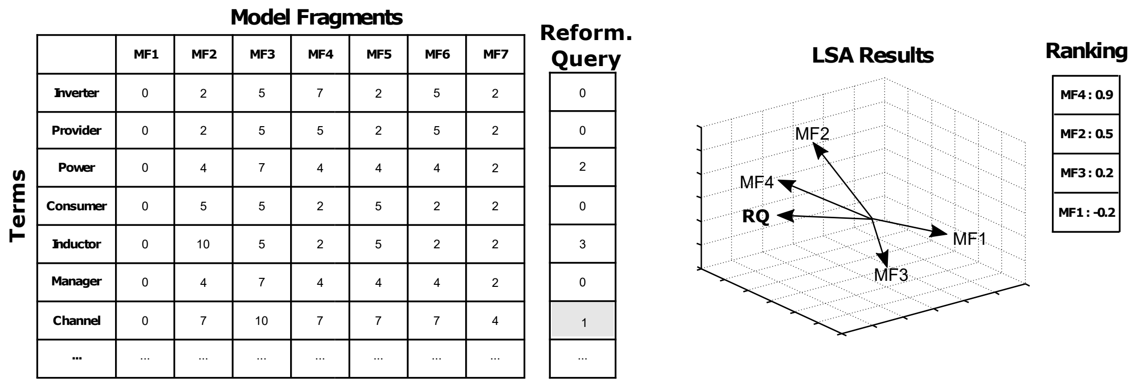
LSA constructs vector representations of a query and a corpus of text documents by encoding them as a term by document co-occurrence matrix, (i.e., a matrix where each row corresponds to terms and each column corresponds to documents, with the last column corresponding to the query). Each cell holds the number of occurrences of a term (row) inside a document or the query (column).

In our work, all documents are model fragments. For each model fragment, texts that represent each element are extracted and then concatenated into a single string. This string is used as a document of the model fragment. For example, if a model fragment specified with the IHDSL contains the element inverter, the document of this model fragment includes all texts that represent the inverter (i.e., the name of the element as well as the values of its specific properties and methods). The query is constructed from the terms that appear in the reformulated query obtained from the feature description provided by the user. The text from the documents (model fragments) and the text from the reformulated query are homogenized by applying well-known Natural Language Processing techniques as follows:

1. The text is tokenized (divided into words). Usually, a white space tokenizer can be applied (which splits the strings whenever it finds a white space), but for some sources of description, more complex tokenizers need to be applied. For instance, when the description comes from documents that are close to the implementation of the product, some words could be using CamelCase naming.
2. The Parts-of-Speech (POS) tagging technique is applied to analyze the words grammatically and to infer the role of each word in the text provided. As a result, each word is tagged, which allows the removal of some categories that do not provide relevant information. For instance, conjunctions (e.g., *or*), articles (e.g., *a*) or prepositions (e.g., *at*) are words that are commonly used and do not contribute relevant information that describes the feature, so they are removed.
3. Stemming techniques are applied to unify the language that is used in the text. This technique consists of reducing each word to its roots, which allows different words that refer to similar concepts to be grouped together. For instance, plurals are turned into singulars (*inductors* to *inductor*) or verbs tenses are unified (*using* and *used* are turned into *use*).

The union of all the keywords extracted from the documents (model fragments) and from the reformulated query are the terms (rows) used by our LSA fitness. The left side of Figure 5 shows an example of the co-occurrence matrix for our running example. Each column is one of the model fragments. The column near to the matrix is the reformulated query in which new terms are added (such as channel as denoted in the shadow cell of the figure) as a result of applying a query expansion technique. Each row is one of the terms extracted from the corpuses of text composed by all of the model fragments and the reformulated query itself (we show the terms before the stemming process to improve readability). Each cell shows the number of occurrences of each of the terms in the model fragments.

Once the matrix is built, we use it as input to obtain the fitness values that assess the relevance of each model fragment in relation to the reformulated query. To do this, we normalize and decompose the matrix into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [27]. Specifically, it is obtained one vector for each model fragment and the reformulated query in order to represent the latent semantics. Once the set of vectors are obtained, a similarity value is calculated for each model fragment as the cosine between its corresponding vector and the vector of the reformulated



Model Fragments

Terms	MF1	MF2	MF3	MF4	MF5	MF6	MF7
Inverter	0	2	5	7	2	5	2
Provider	0	2	5	5	2	5	2
Power	0	4	7	4	4	4	2
Consumer	0	5	5	2	5	2	2
Inductor	0	10	5	2	5	2	2
Manager	0	4	7	4	4	4	2
Channel	0	7	10	7	7	7	4
...	...	...	...	...	...	...	...

Reform. Query
0
0
2
0
3
0
1
...

LSA Results

Ranking
MF4 : 0.9
MF2 : 0.5
MF3 : 0.2
MF1 : -0.2

Figure 5: Fitness Operation via Latent Semantic Analysis (LSA)

query. The similarity value that is calculated to each model fragment is the fitness value, which is between -1 and 1.

A three-dimensional graph of the LSA results is shown in Figure 5. The graph shows the representation of vectors, which are labeled with letters that represent the names of the model fragments (e.g., MF1) and the reformulated query (RQ). The right part of Figure 5 shows the similarity values (from highest to lowest value) of each model fragment, which are calculated by computing the cosine between the associated vector and the vector of the reformulated query.

4. Evaluation

We conducted an evaluation to compare the results provided by our approach with the four automatic query reformulation techniques that were presented in Section 2 (Query reduction, Rocchio query expansion, RSV query expansion and Dice query expansion) in two industrial model-based product families from two of our partners: BSH, the leading manufacturer of home appliances in Europe; and CAF, an international provider of railway solutions all over the world.

4.1. Definition

There are several aspects that we want to evaluate. First, we want to investigate the performance in terms of recall and precision of locating features in industrial model-based product families using the query as it is and using four different automatic query reformulation strategies (Query reduction, Rocchio query expansion, RSV query expansion and Dice query expansion). Second, we want to establish which strategy works better. Our conjecture is based on prior FL works [7, 16, 10, 11], which concluded that reformulation strategies (query expansion and query reduction) can locate more relevant code. For example, query expansion could improve up to 20% [11]. Hence, third, we compare the use of the query as it is with the four query reformulation strategies to establish whether query reformulation strategies should be considered to be applied to locate features in models. In order to address these issues, we formulate the three following questions:

RQ$_1$: *What is the performance locating features using the query as it is and using the four different automatic query reformulation techniques in two industrial model-based product families?*

RQ$_2$: *Are there significant differences in performance using the query as it is and using the query reformulation techniques?*

RQ$_3$: *Does the automatic query reformulation techniques perform better than using the query as it is in models?*

Answering RQ$_1$ allows us to inform about the performance results in terms of recall and precision of query reformulation techniques, which perform the best retrieving code but they have been never used to locate features in industrial model-based families. A positive answer for RQ$_2$ implies that there are statistically significant differences in performance using the query as it is and the query reformulation techniques. Answering RQ$_3$ allows us to determine whether query reformulation techniques perform better than using the query as it is, and to assess the magnitude of improvement.

4.2. Data set

The data set is provided by two industrial case studies from two of our partners: BSH, the leading manufacturer of home appliances in Europe; and CAF, an international provider of railway solutions all over the world. The industrial partners already had a product family where the approved feature realizations and the feature descriptions where known. It is important to note that extracting the feature descriptions from documentation not structured in features can be tedious and error-prone. However, that challenge is out of the scope of our current work.

4.2.1. BSH

The first case study is BSH (already introduced in Section 3 as the running example). BSH provided us with 46 induction hob models where, on average, each product model is composed of more than 500 elements. In addition, BSH provided us with 96 different features that can be part of a particular product model. Those features correspond to products that are currently being sold or will be released to the market in the near future. For each of the 96 features, BSH provided us the following: the set of product models (the product model family), the feature description, a set of relevant model fragments where the feature is used (this information is obtained from the product models), and the approved feature realization (the model fragment that realize the feature). Next, we create a test case for each feature in which we check that there are model fragments for each feature description, and the model fragments were in the product models provided for the evaluation in order to avoid errors that could affect the results.

4.2.2. CAF

The second case study is CAF. Their trains can be seen all over the world and in different forms (regular trains, subway, light rail, monorail, etc.). A train unit is furnished with multiple pieces of equipment through its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their aim is to carry out specific tasks for the train. Some examples of these devices are: the traction equipment, the compressors that feed the brakes, the pantograph that harvests power from the overhead wires, or the circuit breaker that isolates or connects the electrical circuits of the train. The control software of the train unit is in charge of making all the equipment cooperate to achieve the train functionality while guaranteeing compliance with the specific regulations of each country.

The DSL of our industrial partner has the required expressiveness to describe the interaction between the main pieces of equipment installed in a train unit. Moreover, this DSL also has the required expressiveness to specify non-functional aspects related to regulation, such as the quality of signals from the equipment or the different levels of installed redundancy.

As an example, the high voltage connection sequence can be described with the DSL. One of the scenarios from this sequence is the one presented when it is initiated, under demand from the train driver, who requests its start through interface devices fitted inside the cabin. The control software is in charge of raising the pantograph to harvest power from the overhead wire and of closing the circuit breaker so the energy can get to converters that adapt the voltage to charge batteries which, in turn, power the traction equipment. Another example of the functionality that the DSL can specify is the coupling between train units. A train unit can physically connect to a second train unit and control it in order to increase its passenger capacity or to rescue the second train unit in case it has suffered any damage while functioning.

CAF provided us with 23 trains where each product model is composed of more than 1200 elements on average. They are built from 121 different features that can be part of a particular product model. For each of the 121 features, CAF provided us with the following: the product model family, the feature description, a set of relevant model fragments where the feature is used, and the approved feature realization. For each feature, we create a test case in which we check that there are model fragments for each feature description, and the model fragments were in product model family provided for the evaluation in order to avoid errors that could affect the results.

4.3. Planning and execution

Figure 6 shows an overview of the process that was planned to evaluate our FL approach using the query as it is (baseline) and using the four query reformulation techniques that were described in Section 2 (Query Reduction, Rocchio Expansion, RSV Expansion, and Dice Expansion). The left part of the figure shows the inputs of the evaluation process, which are the data set provided by our industrial partners: the product family, the feature descriptions, the relevant model fragments for each feature (to generate term suggestions for the query expansion), and the approved feature realization for each feature. After the collection of the input data, we perform the following steps:

1. Document corpus and oracle creation. We build the document corpus by extracting the terms found in the names and values of the properties and methods of each product model of the product model family as a separate document. We then homogenized

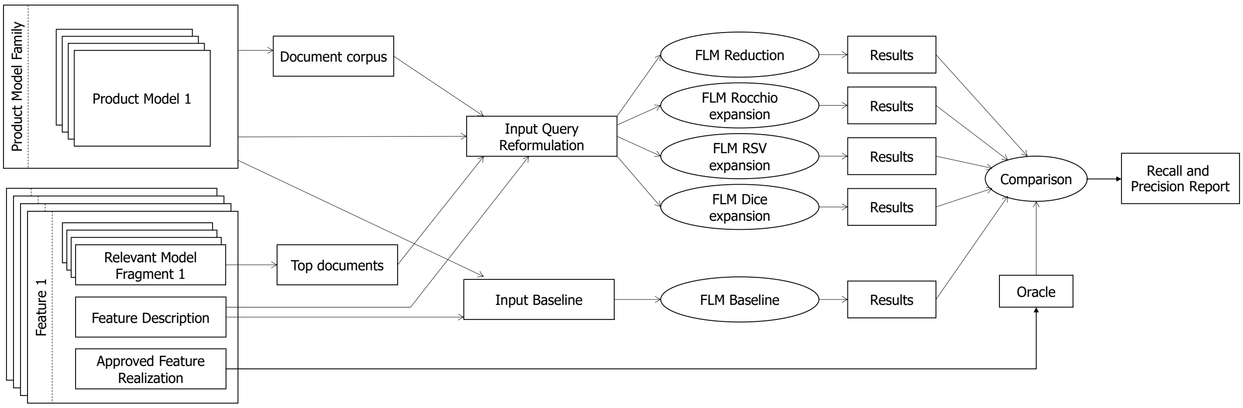


Figure 6: Evaluation execution and measurement

the terms from the documents using the well-known Natural Language Processing techniques (identifier splitting, stop words removal, and stemming) as described in Section 3.3.

After, it is taken the approved feature realization provided by our industrial partners as the oracle, which is the model fragment that realizes a target feature. The oracle will be considered the ground truth and will be used to measure and compare the results of the different techniques as the right side of Figure 6 shows.

2. Feature location and performance measurement. We perform the feature location for each feature description and its reformulations. On the one hand, we perform the feature location for the feature description (baseline) taking as input the product model family and the homogenized terms of each feature description. On the other hand, we perform the feature location for each query reformulation (reduction, Rocchio expansion, RSV expansion, and Dice expansion) taking as input the document corpus, the homogenized terms of each feature description and the top documents for each feature. The top documents are relevant model fragments provided by our industrial partners in which their terms are extracted and homogenized in order to generate term suggestions for expanding the feature description (i.e., query). In order to homogenize the terms, well-known Natural Language Processing techniques (identifier splitting, stop words removal, and stemming) are used as described in Section 3.3. In order to generate term suggestions, the top documents are five relevant model fragments. Also, when expanding the query, we considered the first 10 term suggestions. These decisions were made based on recommendations found in the domain literature [13].

Once the feature location is performed, a ranking of model fragments is obtained as result for each feature description and for each reformulation. Next, we compare the best solution of each ranking (i.e., the model fragment that is in the first position since it has the highest similarity value with the input feature description or reformulation) with the oracle in order to obtain the performance measurement in terms of precision and recall values.

Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle), and recall measures the number of elements of the solution that are retrieved by the proposed solution. A measure that combines both recall and precision is the harmonic mean of precision and recall, which is called the F-measure.

The recall and precision were calculated as follows:

$$Precision = \frac{SolutionElements \cap OracleElements}{SolutionElements}$$

$$Recall = \frac{SolutionElements \cap OracleElements}{OracleElements}$$

The F-measure that combines recall and precision was calculated as follows:

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

To calculate the precision and recall, we need to compute the true positives (TP); the number of elements in the solution that are actually correct according to the ground truth (the oracle), i.e., the number of elements that are present in both the solution and the ground truth. The precision is calculated by dividing the TP by the total number of elements in the solution. The recall is calculated by dividing the TP by the total number of elements in the ground truth.

In our case, each feature is a model fragment composed of a subset of the model elements that are present in the product model (where the feature is being located). Since, the granularity will be at the level of model elements, each model element that is present in both the solution feature candidate and the located feature from the oracle will be a TP.

Recall values can range between 0% (which means that no single model element from the realization of the feature obtained from the oracle is present in any of the model fragments of the feature candidate) to 100% (which means that all the model elements from the oracle are present in the feature candidate).

Precision values can range between 0% (which means that no single model fragment from the feature candidate is present in the realization of the feature obtained from the oracle) to 100% (which means that all the model fragments from the feature candidate are present in the feature realization from the oracle). A value of 100% precision and 100% recall implies that both feature realizations are the same.

In our work, we evaluate the query reformulation for feature location in models by means of two industrial model families. In these model families, our industrial partners do know the mapping of the features, and we take advantage of that knowledge to use it as the oracle to evaluate the performance of the techniques. It is important to clarify that the techniques do not take as input any information from the mapping (oracle). The mapping (oracle) is only used to compare the quality of the results of each technique (see Figure 6, where the mapping is denoted as Approved Feature Realization). Therefore, the techniques face an input where there is no information about the mapping, as it happens when a user wants to locate a feature where the mapping of the feature is not known.

We are not the first researchers to use this oracle schema. The following paper [28] discusses the use of the oracle, which has been extensively used by the research community in conjunction with the measurements of precision and recall.

***3.Answering RQ*₁.** For the BSH case study, we executed our approach 30 independent runs for each of the 96 test cases and for each technique (as suggested by [29]), i.e., 96 (features) x 5 (techniques: the baseline and the four query reformulations) x 30 repetitions = 14400 independent runs. For the CAF case study, we executed our approach 30 independent runs for each of the 121 test cases for each technique, i.e., 121 (features) x 5 (techniques) x 30 repetitions = 18150 independent runs. The sum for the two case studies presented gives a total of 32550 independent runs.

We record the mean values and standard deviations for precision, recall and F-measure for each case study and technique (the baseline and the four query reformulations).

***4. Answering RQ*₂.** To properly compare our baseline approach and automatic query reformulation techniques, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [29] to provide formal and quantitative evidence (statistical significance) that the baseline and the query reformulation techniques do in fact have an impact on the comparison metrics (i.e., that the differences in the results were not obtained by mere chance).

To enable statistical analysis, all of the techniques should be run a large enough number of times (in an independent way) to collect information on the probability distribution for each technique. A statistical test should then be run to assess whether there is enough empirical evidence to claim (with a high level of confidence) that there is a difference between the two techniques (e.g., A is better than B). In order to do this, two hypotheses, the null hypothesis H_0 and the alternative hypothesis H_1 are defined. The null hypothesis H_0 is typically defined to state that there is no difference among the techniques, whereas the alternative hypothesis H_1 states that at least one technique differs from another. In such a case, a statistical test aims to verify whether the null hypothesis H_0 should be rejected.

The statistical tests provide a probability value, $p-value$. The $p-value$ obtains values between 0 and 1. The lower the $p-value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p-value$ under 0.05 is statistically significant [29], and so the hypothesis H_0 can be considered false.

The test that we must follow to obtain a $p-value$ depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. Although there are several tests for analyzing this kind of data, we follow the Quade test to obtain a p-value that verifies whether the statistical hypothesis H_0 can be considered false (i.e., at least one technique differs from another). We made this decision based on the following recommendations found in the literature [30, 31]: the Quade test shows that it is more powerful than others when working with real data [30], and the Quade test has shown better results than the others when the number of techniques to be compared is low (no more than 4 or 5 techniques) [31].

In addition, the performance of the baseline should be individually compared against all other techniques to determine whether statistically significant differences exist among the results of a specific pair of techniques. In order to do this, we perform an additional post hoc analysis, which performs a pair-wise comparison between the baseline and each query reformulation technique.

***5. Answering RQ*₃.** When comparing techniques with a large enough number of runs,

Table 2: Mean values and standard deviations for Precision, Recall, and F-measure in the baseline and the reformulation techniques in the two case studies

	Precision ± (σ)		Recall ± (σ)		F-measure ± (σ)	
Technique	BSH	CAF	BSH	CAF	BSH	CAF
Baseline	83.26 ± 8.70	83.75 ± 7.98	81.54 ± 9.20	81.96 ± 7.57	81.89 ± 6.71	82.43 ± 5.46
Query Reduction	80.51 ± 9.75	80.95 ± 9.69	45.87 ± 14.82	46.88 ± 12.30	56.82 ± 13.87	58.24 ± 10.57
Rocchio Expansion	30.32 ± 14.72	32.39 ± 12.72	79.60 ± 10.09	81.52 ± 9.82	41.63 ± 15.47	44.74 ± 13.72
RSV Expansion	39.44 ± 14.51	40.86 ± 14.38	81.76 ± 9.29	82.77 ± 8.97	51.62 ± 14.09	53.06 ± 13.79
Dice Expansion	31.87 ± 13.99	32.12 ± 13.12	80.41 ± 7.63	81.62 ± 9.43	43.68 ± 15.37	44.45 ± 14.07

statistically significant differences can be obtained even if they are so small as to be of no practical value [29]. Then it is important to assess if a technique is statistically better than another and to assess the magnitude of the improvement. *Effect size* measures are needed to analyze this.

For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [32, 33]. $\hat{A}_{12}$ measures the probability that running one technique yields higher values than running another technique. If the two techniques are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results in 70% of the runs with the first of the pair of techniques that have been compared, and $\hat{A}_{12} = 0.3$ means that we would obtain better results in 70% of the runs with the second of the pair of techniques that have been compared. Thus, we record an $\hat{A}_{12}$ value for the baseline with any of the query reformulation techniques.

5. Results

In this section, we present the results obtained for each case study in the baseline and the four automatic query reformulations to answer each research question.

5.1. Research Question 1

Table 2 shows the mean values of precision, recall, and F-measure for the features (96 features in BSH and 121 features in CAF) and the 30 repetitions.

RQ$_1$ answer. The baseline obtains the best result for precision, providing a precision value of 83.26% in the BSH case study and a precision value of 83.75% in the CAF case study, which outperforms the worst precision value about 52%. The RSV expansion query reformulation technique obtains the best result for recall (81.76% in the BSH case study and 82.77% in the CAF case study), whereas the second highest result for recall is obtained by the baseline (81.54% in the BSH case study and 81.96% in the CAF case study). In terms of precision and recall, the results are slightly higher in the CAF case study.

5.2. Research Question 2

The $p - Values$ and statistics of the Quade test are shown in Table 3. Since the $p - Values$ shown in this table are smaller than $2x10^{-16}$ in all cases, we reject the null hypothesis. Consequently, we can state that there exist differences in the techniques for all the performance indicators (precision and recall).

Table 3: Quade test statistic and $p - Values$

	BSH		CAF	
	Precision	Recall	Precision	Recall
$p - Value$	$\ll 2.2x10^{-16}$	$\ll 2.2x10^{-16}$	$\ll 2.2x10^{-16}$	$\ll 2.2x10^{-16}$
Statistic	129.83	55.06	159.96	68.98

However, the Quade test does not determine whether there are statistically significant differences between the baseline and a reformulation technique. Hence, the performance of the baseline should be individually compared against all other reformulations by performing an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each algorithm, determining whether statistically significant differences exist. To do this, we apply the Holm Post Hoc analysis as suggested by Garcia et al. [30]. This analysis obtains a $p - Value$ for each pair-wise comparison. It is accepted by the research community that a $p - value$ under 0.05 is statistically significant [29].

Table 4 shows the $p - Values$ of Holm's post hoc analysis for pair-wise comparisons between the performance indicators (precision and recall) of baseline and each reformulation technique in the BSH and CAF case studies. The results show that all pair-wise comparisons have statistically significant differences in one of the performance indicators because there are values smaller than the corresponding significance threshold value (0.05).

Table 4: Holm's post hoc $p - Values$

	BSH		CAF	
	Precision	Recall	Precision	Recall
Base vs Query Reduction	0.84	$\ll 2x10^{-16}$	1.00	$\ll 2x10^{-16}$
Base vs Rocchio Expansion	$\ll 2x10^{-16}$	1.00	$\ll 2x10^{-16}$	1.00
Base vs RSV Expansion	$\ll 2x10^{-16}$	1.00	$\ll 2x10^{-16}$	1.00
Base vs Dice Expansion	$\ll 2x10^{-16}$	0.97	$\ll 2x10^{-16}$	1.00

$\mathbf{RQ_2}$ answer. We determine that there are significant differences in all performance indicators (precision and recall) among the techniques in the two case studies. Specifically, there are significant differences for recall between the baseline and the query reduction technique, whereas there are significant differences for precision between the baseline and each query expansion technique.

5.3. Research Question 3

Table 5 shows the values of the effect size statistics between the baseline and each query reformulation technique. The $\hat{A}_{12}$ values for precision in the BSH and CAF case studies show the largest differences, with values up to 0.9985 and 0.99993, respectively. This indicates that the baseline outperforms the Rocchio, RSV and Dice query expansion techniques for precision with a pronounced superiority, and the baseline slightly outperforms the query reduction technique. With regard to recall, the baseline shows a pronounced superiority with the query reduction technique (0.9829 in BSH and 0.9917 in CAF), whereas the values are near to the equivalent value of 0.5 in the two case studies for the query expansion techniques (Rocchio, RSV and Dice). Overall, these results confirm that the use of the query reformulation techniques for locating model fragments does not have as much impact as locating code [7], especially on the results for precision.

Table 5: $\hat{A}_{12}$ statistic for each pair of algorithms

	BSH		CAF	
	Precision	Recall	Precision	Recall
Base vs Query Reduction	0.5804	0.9829	0.5739	0.9917
Base vs Rocchio Expansion	0.9978	0.5439	0.9993	0.5007
Base vs RSV Expansion	0.9897	0.4976	0.9939	0.4571
Base vs Dice Expansion	0.9985	0.5455	0.9993	0.4931

RQ$_3$ answer. We conclude that the baseline achieves statistically significant better results compared to each query reformulation technique. Hence, the expansion/reduction reformulation techniques do not perform better than using the query as it is in models.

6. Discussion

Although query reformulation techniques have traditionally been applied to code (where satisfactory results have been obtained), their application to models for feature location is novel. Since models and code are considerably different, it is reasonable to think that these techniques are not going to work. However, there are experiences of text-based techniques traditionally applied to code that have provided satisfactory results in models. For example, natural language processing techniques [34], or information retrieval (Latent Semantic Indexing or Formal Concept Analysis) [35].

In general, query reformulation techniques work with terms that fit the vocabulary of the software artifact. In the same way as variable names or methods encode domain knowledge in code, the names of model elements and their properties also encode domain knowledge.

The differences between code and models may discourage the application of query reformulation techniques in models. However, both (1) the previous experiences of techniques that are shifted from code to models, and (2) the existence of terms with vocabulary of the software artifact that can be also found in models, encourages to explore the possibility of applying query reformulation techniques to models.

The results presented in this paper show that the usage of query reduction or query expansion techniques lead to make the results worse in models. Next, we discuss why this is the case, and how query reformulation may be adjusted for models.

In comparison to code, models raise the abstraction level using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain [36]. Since models are not contaminated with implementation details, one might think that reformulation techniques should work even better than when they are applied to code. The results of reformulations applied to models should include terms related with the domain (e.g., inductor) instead of terms related to implementation details (e.g., the auxiliary variable named tempInductor).

However, our analysis of the results reveals a turn to the worst when reformulation is applied to models. When expansion techniques are applied, the resulting terms (proposed terms to expand the feature description) are always terms from the metamodel. A term such as cutoff (an enumeration of safety values to prevent damage of electronic components) repetitively appears as part of the results of the reformulation. However, these metamodel terms encode very generic domain knowledge and either they are already present in the feature description, or they are not relevant at all for that particular feature description.

When the reduction technique is applied, the results (proposed terms to be removed from the feature description) also include terms from the metamodel such as inverter. Inverter is one of the most popular terms in the induction hob domain and consequently is selected by the reduction technique. However, if the feature description includes the term inverter, removing it leads to a worsening of the results because model elements related to inverter are present in the solution according to the oracle.

There is a characteristic of the models that is not present in code, namely, the domain information is embedded at two levels: the model and the metamodel. The cutoff property of the power managers is defined at metamodel level, but that property is also present at model level. In fact, the value of each power manager is established at model level.

Current reformulation techniques do not take into account the differences between model level and metamodel level. Given a term from a relevant model fragment, reformulation techniques give it the same treatment despite its model or metamodel origin.

Our work reveal that query reformulation techniques should be adjusted for models by considering the differences between model level and metamodel level. It is possible to think that an adjust is to ignore all terms that come from the metamodel. However, we suggest that the union of the property value (from the model) to the property name (from the metamodel) should be performed for the calculations of the reformulations, but only the property value (without the property name) should be added to the resulting list of terms. Experiments should be conducted to establish the best adjust. In fact, this constitutes our future work.

7. Threats of validity

We use the classification of threats of validity of [37, 38], which distinguishes four aspects of validity to acknowledge the limitations of our evaluation.

Construct validity: This aspect of validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind and what is investigated according to the research questions. To minimize this risk, our evaluation is performed using three measures: precision, recall and F-measure. These measures are widely accepted in the software engineering research community [6]. Another threat is the not accounting for random variation. To address this threat, we considered 30 independent runs of our approach for each test case as suggested by [29].

Internal Validity: This aspect of validity is of concern when causal relations are examined. We used an oracle (obtained from our industrial partners and considering the ground truth) to evaluate our approach using feature descriptions as queries where the expected solution was known beforehand. By doing so, we were able to compute the recall, precision and F-measure for the feature candidates rankings provided by the approach. However, when applying the presented approach to locate features (and thus not having an oracle), the approach should be used iteratively in order to refine the query.

Another threat of this type that has been identified is the poor parameter settings. As suggested by Arcuri and Fraser [21], default values are good enough to measure the performance. However, we plan to evaluate all the parameters of our algorithm in future iterations of our work.

With regard to the number of documents and terms used to expand the query, we used the values of 5 and 10, respectively as recommended in the literature [13]. However, we do not know at this stage how using different values would impact the results. We also do not know how the results would change if we modify the value of appearance (25% as previously used in software engineering [12]) of the terms of the query in the documents of the corpus to reduce the terms of the query.

External Validity: This aspect of validity is concerned with to what extent it is possible to generalize the finding, and to what extent the findings are of relevance for other cases. In order to mitigate this threat, we selected two model-based families of software products from different domains since our approach has been designed to be applied not only to the domains of our industrial partners but also, to different domains. The requisites to apply our approach are that the set of models where features have to be located conform to MOF (the OMG metalanguage for defining modeling languages), and the query must be provided as a textual description. Furthermore, the fitness function can also be applied to any MOF-based model. The text elements associated to the models are extracted automatically by the approach using the reflective methods provided by the Eclipse Modeling Framework. That is, there is no need of knowledge about the domain of application in order to extract the relevant terms.

Query reformulation techniques can only work if the original query is reasonably strong to retrieve at least some of the relevant documents [16]. As occurs in other works [16, 17], results depend on the quality of the queries. Poor queries assign high rank to irrelevant model fragments. If irrelevant model fragments are obtained in the solution, the engineers can consider these solutions as a starting point from where solutions can be manually refined, or they may refine the query to automatically obtain different solutions. Hence, 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 particular for each domain, but as long as both elements are built using the same terminology the LSA will work. Eventually, some tweaks can be applied to narrow the gap between both elements (different tokenizers, stemming or POS tagging techniques). For instance, the naming conventions used by companies for model elements, properties and functions can follow different formats, but the approach can be tailored to handle them. In our case studies some model elements follow the CamelCase convention while others follow the Underscore convention. To address that, we applied different tokenizers in order to obtain the terms of the model fragments properly before they are homogenized.

Hence, despite our approach can be applied to locate features on MOF-based model from different domains, our approach should be applied to other domains before assuring its generalization.

Reliability: This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. To reduce this threat, the feature descriptions and the product family are provided by our industrial partners not involved in this research.

8. Related work

Many feature location approaches that have been proposed to address more than twenty tasks in software engineering (e.g., concept/feature/concern location, code retrieval and reuse, etc.) by finding relevant code taking textual information as input [1]. For example, Cavalcanti et al. [39] used IR techniques to assign change requests in software maintenance or evolution tasks based on context information. Kimmig et al. [40] proposed an approach for translating NL queries to concrete parameters of the Eclipse JDT code query engine.

Other approaches 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. [41] proposed a code search approach, which incorporates user feedback to refine the query. Zou et al. [42] proposed a re-ranking approach to refine search results by investigating the "answer style" of software questions with different interrogatives.

Recently, other approaches propose to automatically reformulate the query by removing words to reduce long queries. For example, Bendersky and Croft [43] propose an approach to reduce long queries by learning and identifying the key concepts. Kumaran and Allan [44] propose an interactive approach with users to completely dropping unnecessary terms from long queries. Kumaran and Carvalho [10] reduce queries by analyzing the most promising subsets of terms from the original query.

By contrast, other approaches propose to automatically expand the query to add words that are either similar or related in some way to the query terms. For example, Marcus et al. [45] expand the query using LSI in order to determine the most similar terms to the query from the source code. Yang et al. [9] use the context in which query words are found in the source code to extract synonyms, antonyms, abbreviations and related words to include them in the reformulated query. Hill et al. [17] also use word context in order to extract possible query expansion terms from the code. Haiduc et al. [7] propose an approach 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.

Some approaches expand the query by adding information from external sources of information such as public repositories [46] or adding semantic similar words from websites [47]. For example, Dietrich et al. [48] improved the efficacy of future queries using feedback captured from a validated set of queries and traceability links. Lv et al. [49] enrich each API with its online documentation to match the query based on text similarity.

Despite the effort of improving the performance of feature location in software engineering by applying IR and automatically reformulating the queries, it has been neglected in models. In contrast, our approach addresses feature location in a model-based family of software products by applying IR, and we test four existing automatic query reformulation techniques for expanding or reducing queries that have been tested to locate features in code but not in models.

9. Concluding remarks

We have tested four existing automatic query reformulation techniques that expand or reduce terms from an incoming feature description to check whether these techniques improve the performance locating features in a model-based family of software products as they do in other software artifacts such as code. To this aim, we have used our FL approach that provides the model fragment from a given product family that realizes the incoming feature description. The requisites to apply our FL approach are that the set of models where features have to be located are based on MOF (the OMG metalanguage for defining modeling languages), and the feature description has to be provided as a textual description. Also, we have implemented the four existing automatic query reformulation techniques to expand or reduce terms.

We have performed the evaluation in two industrial domains: a model-based family of firmwares for induction hobs, and a model-based family of PLC software to control trains. For each case study, we have recorded the performance of the feature description (query) and each reformulated query in terms of precision and recall. We have compared the results provided by our FL approach using the query with the results provided by the four reformulation techniques by means of statistical analysis.

Although prior FL works concluded that reformulation strategies (query expansion and query reduction) can improve up to 20% the location of code, our results reveal that these reformulation strategies cannot be used in models as they are in code since they achieved statistically significant worse results. Hence, our work suggests four directions that require more experimentation to improve the location of features in models by means of automatic query reformulations:

- The number of terms used to expand the query, and the value of appearance to reduce the terms of the query.
- The number of relevant documents to expand the query.
- The use of other documents to expand or reduce the query.
- Reconsideration of query reformulation techniques to be applied in models.

We believe that query reformulation techniques should be reconsidered taking into account the rules that exist in the literature to extract conceptual models from requirements [50]. These rules can contribute to the automatic query reformulation techniques with a classification of the terms according to the role that they can play in the models, which could help to propose new criteria of expansion or reduction of queries. In fact, this constitutes our future work.

To conclude, thanks to the retrieved results, we found out that we should not get carried away by inertia and use automatic query reformulation techniques for feature location in models by now as it is done in code. In fact, using these techniques by inertia may lead to worsening the results.

Acknowledgements

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

References

[1] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, Feature location in source code: a taxonomy and survey., Journal of Software: Evolution and Process 25 (2013) 53–95.

[2] M. M. Lehman, J. F. Ramil, G. Kahen, A paradigm for the behavioural modelling of software processes using system dynamics, 2001.

[3] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, A. Rummler, An exploratory study of information retrieval techniques in domain analysis, in: 12th International Software Product Line Conference, 2008, pp. 67–76. doi:10.1109/SPLC.2008.18.

[4] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman, Indexing by latent semantic analysis, Journal of the American Society for Information Science 41 (1990) 391–407.

[5] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, J. Mach. Learn. Res. 3 (2003) 993–1022.

[6] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Inc., 1986.

[7] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, T. Menzies, Automatic query reformulations for text retrieval in software engineering, in: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 842–851.

[8] E. Hill, L. Pollock, K. Vijay-Shanker, Improving source code search with natural language phrasal representations of method signatures, in: Automated Software Engineering (ASE), 2011, pp. 524–527. doi:10.1109/ASE.2011.6100115.

[9] J. Yang, L. Tan, Inferring semantically related words from software context, in: Mining Software Repositories (MSR), 2012, pp. 161–170. doi:10.1109/MSR.2012.6224276.

[10] G. Kumaran, 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, SIGIR '09, ACM, New York, NY, USA, 2009, pp. 564–571. doi:10.1145/1571941.1572038.

[11] X. A. Lu, R. B. Keefer, Query expansion/reduction and its impact on retrieval effectiveness, in: Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994, 1994, pp. 231–240.

[12] G. Gay, S. Haiduc, A. Marcus, T. Menzies, On the use of relevance feedback in ir-based concept location., in: ICSM, IEEE Computer Society, 2009, pp. 351–360. URL: http://dblp.uni-trier.de/db/conf/icsm/icsm2009.html#GayHMM09.

[13] C. Carpineto, G. Romano, A survey of automatic query expansion in information retrieval, ACM Comput. Surv. 44 (2012) 1:1–1:50.

[14] G. Sridhara, E. Hill, L. L. Pollock, K. Vijay-Shanker, Identifying word relations in software: A comparative study of semantic similarity tools., in: R. L. Krikhaar, R. LÃďmmel, C. Verhoef (Eds.),

ICPC, IEEE Computer Society, 2008, pp. 123–132. URL: http://dblp.uni-trier.de/db/conf/iwpc/icpc2008.html#SridharaHPV08.

[15] G. Salton, The SMART Retrieval System—Experiments in Automatic Document Processing, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[16] B. Sisman, A. C. Kak, Assisting code search with automatic query reformulation for bug localization, in: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013, 2013, pp. 309–318. doi:10.1109/MSR.2013.6624044.

[17] E. Hill, L. Pollock, 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, ICSE '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 232–242. URL: http://dx.doi.org/10.1109/ICSE.2009.5070524. doi:10.1109/ICSE.2009.5070524.

[18] L. Davis, Handbook of genetic algorithms 115 (1991).

[19] Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, A. Svendsen, Adding standardized variability to domain specific languages, in: Software Product Line Conference, 2008. SPLC '08. 12th International, 2008, pp. 139–148. doi:10.1109/SPLC.2008.25.

[20] A. S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable product line configuration: A straw to break the camel's back, in: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, ASE'13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 465–474. URL: https://doi.org/10.1109/ASE.2013.6693104. doi:10.1109/ASE.2013.6693104.

[21] A. Arcuri, G. Fraser, Parameter tuning or default values? an empirical investigation in search-based software engineering, Empirical Software Engineering 18 (2013) 594–623.

[22] A. Kotelyanskii, G. M. Kapfhammer, Parameter tuning for search-based test-data generation revisited: Support for previous results, in: 2014 14th International Conference on Quality Software, 2014, pp. 79–84. doi:10.1109/QSIC.2014.43.

[23] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (1998) 259–284.

[24] M. Revelle, B. Dit, D. Poshyvanyk, Using data fusion and web mining to support feature location in software, in: IEEE 18th International Conference on Program Comprehension (ICPC), 2010, pp. 14–23. doi:10.1109/ICPC.2010.10.

[25] D. Liu, A. Marcus, D. Poshyvanyk, V. Rajlich, Feature location via information retrieval based filtering of a single scenario execution trace, in: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07, ACM, New York, NY, USA, 2007, pp. 234–243. URL: http://doi.acm.org/10.1145/1321631.1321667. doi:10.1145/1321631.1321667.

[26] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval, IEEE Transactions on Software Engineering 33 (2007) 420–432.

[27] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (1998) 259–284.

[28] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The oracle problem in software testing: A survey, IEEE Transactions on Software Engineering 41 (2015) 507–525.

[29] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Softw. Test. Verif. Reliab. 24 (2014) 219–250.

[30] S. García, A. Fernández, J. Luengo, 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. 180 (2010) 2044–2064.

[31] W. Conover, Practical nonparametric statistics, Wiley series in probability and statistics, 3. ed ed., Wiley, New York, NY [u.a.], 1999.

[32] A. Vargha, 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 25 (2000) 101–132.

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

[34] R. Lapeña, J. Font, O. Pastor, C. Cetina, Analyzing the impact of natural language processing over feature location in models, in: Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2017, Vancouver, BC, Canada, October 23-24, 2017, 2017, pp. 63–76.

[35] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature location in models through a genetic algorithm driven by information retrieval techniques, in: ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS 2016, 2016.

[36] B. Selic, The pragmatics of model-driven development, IEEE Softw. 20 (2003) 19–25.

[37] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical software engineering 14 (2009) 131–164.

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

[39] Y. a. C. Cavalcanti, I. d. C. Machado, P. A. d. M. S. Neto, E. S. de Almeida, S. R. d. L. Meira, Combining rule-based and information retrieval techniques to assign software change requests, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, ACM, New York, NY, USA, 2014, pp. 325–330. URL: http://doi.acm.org/10.1145/2642937.2642964. doi:10.1145/2642937.2642964.

[40] M. Kimmig, M. Monperrus, M. Mezini, Querying source code with natural language, in: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, 2011, pp. 376–379. doi:10.1109/ASE.2011.6100076.

[41] S. Wang, D. Lo, 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, ASE '14, 2014, pp. 677–682. URL: http://doi.acm.org/10.1145/2642937.2642947. doi:10.1145/2642937.2642947.

[42] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, L. Zhang, Learning to rank for question-oriented software text retrieval, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2015, pp. 1–11.

[43] M. Bendersky, W. B. Croft, Discovering key concepts in verbose queries, in: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '08, ACM, New York, NY, USA, 2008, pp. 491–498. URL: http://doi.acm.org/10.1145/1390334.1390419. doi:10.1145/1390334.1390419.

[44] G. Kumaran, J. Allan, Effective and efficient user interaction for long queries, in: SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, 2008, pp. 11–18. URL: http://portal.acm.org/citation.cfm?id=1390339. doi:http://doi.acm.org/10.1145/1390334.1390339.

[45] A. Marcus, A. Sergeyev, V. Rajlich, J. I. Maletic, An information retrieval approach to concept location in source code, in: Proceedings of the 11th Working Conference on Reverse Engineering, WCRE '04, Washington, DC, USA, 2004, pp. 214–223. URL: http://dl.acm.org/citation.cfm?id=1038267.1039053.

[46] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, M. Mirakhorli, On-demand feature recommendations derived from mining public product descriptions, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, 2011, pp. 181–190. URL: http://doi.acm.org/10.1145/1985793.1985819. doi:10.1145/1985793.1985819.

[47] Y. Tian, D. Lo, J. Lawall, Automated construction of a software-specific word similarity database, in: IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014, pp. 44–53. doi:10.1109/CSMR-WCRE.2014.6747213.

[48] T. Dietrich, J. Cleland-Huang, Y. Shin, Learning effective query transformations for enhanced requirements trace retrieval, in: 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), 2013, pp. 586–591. doi:10.1109/ASE.2013.6693117.

[49] F. Lv, H. Zhang, J. g. Lou, S. Wang, D. Zhang, J. Zhao, Codehow: Effective code search based on API understanding and extended boolean model, in: Automated Software Engineering (ASE2015), 2015.

URL: http://research.microsoft.com/apps/pubs/default.aspx?id=259612.

[50] T. Yue, L. C. Briand, Y. Labiche, atoucan: An automated framework to derive uml analysis models from use case models, ACM Trans. Softw. Eng. Methodol. 24 (2015) 13:1–13:52.