Empowering the Human as the Fitness Function in Search-Based Model-Driven Engineering

Francisca Pérez, Jaime Font, Lorena Arcega, Carlos Cetina

Abstract—In Search-Based Software Engineering, more than 100 works have involved the human in the search process to obtain better results. However, the case where the human completely replaces the fitness function remains neglected. There is a good reason for that; no matter how intelligent the human is, humans cannot assess millions of candidate solutions as heuristics do. In this work, we study the influence of using the Human as the Fitness Function (HaFF) on the quality of the results. To do that, we focus on Search-Based Model-Driven Engineering (SBMDE) because inspecting models should require less human effort than inspecting code thanks to the abstraction of models. Therefore, we analyze the impact of HaFF in a real-world industrial case study of feature location in models. Furthermore, we also consider a reformulation operation (replacement) in the evaluation because a recent work reported that this operation significantly reduces the number of iterations required in comparison to the widespread crossover and mutation operations. The combination of HaFF and the reformulation operation (HaFF_R) improves the results of the best baseline by 0.15% in recall and 14.26% in precision. Analyzing the results, we learned how to better leverage HaFF_R, which increased the improvement with regard to the best baseline to 1.15% in recall and 20.05% in precision. HaFF_R significantly improves precision because humans are immune to the main limitations of the baselines: vocabulary mismatch and tacit knowledge. A focus group confirmed the acceptance of HaFF. These results are relevant for SBMDE because feature location is one of the main activities performed during maintenance and evolution. Our results, and what we learned from them, can also motivate and help other researchers to explore the benefits of HaFF. In fact, we provide a guideline that further discusses how to apply HaFF to other software engineering problems.

Index Terms—Model-Driven Engineering, Search-Based Software Engineering, Automatic Query Reformulations, Interactive SBSE

1 Introduction

In the last few years, great interest has emerged from the application of search-based optimization to software engineering, an area known as Search-Based Software Engineering (SBSE) [1], [2]. Only three key ingredients are needed to apply SBSE: 1) a representation (encoding) of the problem (e.g., using a bit string); 2) the definition of the set of operations (e.g., mutation); and 3) the definition of the fitness function (e.g., similarity to the input query). Then, candidate solutions (which are encoded following the representation chosen) are evolved (by applying the operations) and are assessed (by the fitness function) in an iterative process until a stop condition is met (e.g., a fixed number of iterations). As a result, optimal (or near-optimal) solutions to the problem are found.

Although SBSE is a means of efficiently exploring the huge search space and the sheer number of potential solutions, some contexts require the human's subjective evaluation to be included in the process in order to obtain the most favorable solutions. This reflects a subfield for SBSE entitled interactive SBSE (iSBSE). A recent survey on iSBSE [3] acknowledges that any attempt to involve the human in the search process with the aim of adapting the results to their preferences can be viewed as iSBSE. This includes the case where the human provides subjective evaluations (e.g., scores) of solutions to complement the fitness function [4], [5], [6], [7], [8], [9], [10].

Nevertheless, the survey does not identify any work that completely replaces the fitness function with the human, i.e., where the Human as the Fitness Function (HaFF) evaluates the candidate solutions. On the one hand, HaFF could improve the performance in terms of the solution quality. On the other hand, in a complex process, human fatigue might prevent the human from replacing the fitness function. All in all, HaFF is still neglected in the context of iSBSE.

In this paper, we evaluate the influence of HaFF in Search-Based Model-Driven Engineering (SBMDE) [11], [12]. SBMDE arises from the relevance of Model-Driven Engineering (MDE) and SBSE [11] to software engineering. In SBMDE, the cornerstone artifacts are the software models, which are much less bound to the underlying implementation and technology and raise the abstraction level using terms that are much closer to the problem domain [13]. Since software models are more abstract than code, the human should not have to evaluate as much information in the solutions, which should be more favorable for replacing the fitness function with the human.

Among the SBMDE maintenance tasks, we focus on Feature Location in Models (FLiM). Feature location can be seen as one of the most frequent maintenance tasks undertaken by developers [14], [15], [16], [17]. FLiM consists in identifying the set of model elements (i.e., model fragment) that is associated with a specific 'feature'. The term 'feature' refers to a specific functionality or characteristic of a product. Fig. 1 shows an example of FLiM. The left part of the figure shows that FLiM takes a set of software models as input to identify the model fragment that is relevant for

• *F. Pérez, J. Font, L. Arcega and C. Cetina are with the SVIT Research Group of Universidad San Jorge, Zaragoza, Spain. E-mail: {mfperez, jfont, larcega, ccetina}@usj.es. C.Cetina is also with University College London, London, United Kingdom.*

Manuscript received December X, 2020; revised .

the feature to be located (model elements 1-3, highlighted in gray in the gure).

The right part of Fig. 1 shows two puzzle pieces that can t the tness function in FLiM. The piece in the upper-right part of the gure shows the tness that is the most popular choice in FLiM [11]. In this piece, a software engineer provides a feature description in natural language that is used as input to the tness function, which assesses the model fragments of the population according to the similarity to the input feature description. The piece in the lower-right part of Fig. 1 shows the approach of our work (HaFF), where the tness function is completely replaced by a software engineer who assesses the model fragments of the population.

Fig. 1. Example of FLiM with two options for the tness function

Despite the abstraction of software models, the most popular FLiM approaches require an average of 347685 iterations [18] with a population of 100 model fragments in software models of industrial dimensions (with more than 1000 model elements in each model). Recently, Pérez et al. [18] proposed that operations leverage the latent semantics of software models instead of randomly generating new candidate solutions as the widespread crossover and mutation operations do. Speci cally, they propose utilizing query reformulation techniques (expansion[19], replacement[20], reduction[21], and selection[22]) as genetic operations. They show that replacement is the reformulation that best leverages the latent semantics of software models. This results in a reduction of iterations (on average from 347685 to 384) for FLiM. These numbers seem far from the recommendations of Takagi [23] for avoiding human fatigue, which are the reduction of both the size of the population and the number of iterations to 10 or 20. Nevertheless, in HaFF, since the human will provide the most intelligent search possible, this might have an effect on the required number of iterations.

To analyze the combination of HaFF with either the widespread crossover and mutation operations (HaFF_CM) or the replacement reformulation operation (HaFF_R), we conducted experiments on a real-world industrial case study of FLiM. The case study is from Construcciones y Auxiliar de Ferrocarriles (CAF) [1], who is a world-leading company that uses software models to generate the rmware (C++) to control the trains that have been manufactured over years. In the evaluation, 29 software engineers from the industrial partner were involved in acting as the tness function.

To assess the performance in terms of solution quality (recall, precision, and F-measure), we compared the results with the oracle provided by the industrial partner (which is considered to be the ground truth). To put the performance of HaFF in perspective and to study the impact on the results, we set two baselines where the tness function analyzes the similarity between the model fragment and the feature description. The rst baseline uses crossover and mutation operations (Baseline_CM), and the second baseline uses the replacement reformulation operation (Baseline_R).

The results show that HaFF_CM fails to beat the baselines. Nevertheless, HaFF_R improves the results of the best baseline (Baseline_R) by 0.15% in recall and 14.26% in precision. We also performed a statistical analysis to provide quantitative evidence of the impact of the results and to show that this impact is signi cant. The signi cant improvement in precision comes from the immunity of humans to previously reported limitations of the baselines: vocabulary mismatch and tacit knowledge [24]. Furthermore, our analysis of the results also reveals that HaFF_R can be further improved (from 0.15% to 1.15% in recall, and from 14.26% to 20.05% in precision) if a self-rated feature familiarity assessment is used to determine when HaFF_R or Baseline_R should be used.

To the best of our knowledge, this is the rst interactive SBMDE work that successfully harnesses HaFF at the industrial scale. Our paper claims that HaFF_R is bene cial in the context of SBMDE, as this work shows in the case of FLiM. Speci cally, we claim that:

- HaFF_R improves the results in the task of FLiM compared to the baselines. In addition, we carried out a focus group interview that con rmed the acceptance of HaFF. This is relevant for the SBMDE community because feature location is an essential task for software maintenance and evolution [14], [15].
- HaFF_R does not achieve the best solutions in all cases. Therefore, HaFF_R should not always be used. We provide a recommendation on when to use HaFF_R. This recommendation boosts HaFF_R performance.
- The reformulation operation enables HaFF_R to improve the results of the best baseline. This operation is not well-known within the SBMDE community because: 1) its bene ts were reported very recently; and 2) crossover and mutation operations have been the predominant choice since 1998 [11]. Our work might motivate other SBMDE researchers to rethink the operations of their approaches. If they move from current random operations to smarter operations (such as reformulation, or their own new operations), they might be able to get the bene ts of HaFF in other software engineering tasks.
- HaFF is generic and can be applied to software artifacts other than just model artifacts. We created a guideline that does the following: further discusses how to apply HaFF to other software engineering

1. www.caf.net/en

problems; describes who should play the role of HaFF and how the solutions should be evaluated; presents possibilities to escalate HaFF to more demanding problems where fatigue appears (hybridize or collaborate); and provides recommendations for types of problems that should be used to continue exploring HaFF. We are aware that models are a more favorable scenario than code for HaFF because of their abstraction. However, other SBSE works target other artifacts (such as requirements or release plans) that might be suitable scenarios for HaFF. Our results can also motivate other SBSE researchers to explore the in uence of HaFF with other artifacts.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents an overview of FLiM. Section 4 describes the evaluation. The results are reported in Section 5 and discussed in Section 6. Section 7 presents the guideline for applying HaFF to other software engineering problems. Section 8 describes the threats to validity. Section 9 concludes the paper.

2 Related work

There is a recent survey [11] that includes SBMDE works from 1998 to 2016. In addition, there is another recent survey that includes iSBSE [3] works from 1999 to 2017. We have updated both surveys to the present day and identi ed the common works. Table 1 shows the resulting works. The third column classi es the type of interaction, the fourth column indicates when the interaction occurs (before, during, or after the optimization process), and the fth column classi es the evaluation mechanism used (if the type of interaction is evaluation). This classi cation was performed following the classi cation for interactive approaches presented in [3].

First, we compared the two surveys [3], [11] to obtain the common works between 1999 and 2016 (see the two works in the upper part of Table 1). Both Ghannem et al. [6] and Amal et al. [9] address model refactoring. Ghannem et al. [6] combine structural similarity and designers' ratings of refactorings to evaluate the refactorings. In the approach of Amal et al. [9], the software engineers manually evaluate the suggested refactorings, and an Arti cial Neural Network uses these training examples to evaluate the refactoring solutions for the remaining iterations. Both approaches involve humans in the evaluation of the solutions; however, neither of them completely replaces the tness function with the human as our work does.

Second, we used the queries presented in the surveys [3], [11] to obtain the research works starting from 2016 and 2017 until August 2020, respectively. Then, we identi ed the common works (see the 22 works in the middle part of the table). Most of the works (54.55%) involve the human in the selection of solutions. The next most common type of interaction is modi cation (31.82%). We focus on the evaluation type. Only 27.27% of the works involve the human in the evaluation. Martinez et al. [25], [26] leverage the user feedback to evaluate user interfaces obtained from a Software Product Line. Kessentini et al. [27] propose an approach for metamodel/model co-evolution where developers can approve, modify, or reject the recommended edit operations.

TABLE 1
Human interaction of related SBMDE works

Interactive SBMDE Work	Year	Human Interaction			Industrial Scale
		Type of interaction	When it occurs	Evaluation mechanism	
Ghannem et al. [6]	2013	Evaluation	During	Scores	7
Amal et al. [9]	2014	Evaluation	During	Fitness value	7
Lin et al. [31]	2016	Selection	After	-	7
Yue et al. [32]	2016	Selection, modi cation	Before	-	7
Van Rooijen and Hamann [33]	2016	Selection	After	-	7
Lu et al. [34]	2016	Selection	Before	-	7
Debreceni [35]	2016	Comparison	After	-	3
Batot and Sahraoui [36]	2016	Modi cation	After	-	7
Fleck et al. [37]	2016	Selection, modi cation	After	-	7
Mart´nez et al. [25]	2017	Evaluation	During	HaFF_CM	7
Gómez-Abajo et al. [38]	2017	Modi cation	After	-	7
Calinescu et al. [39]	2017	Selection	After	-	7
Kessentini et al. [27]	2018	Modi cation, evaluation	During	Reward and penalization	7
Mart´nez et al. [26]	2018	Evaluation	During	Scores	7
Marculescu et al. [28]	2018	Evaluation	During	Weights	3
Kolchin [40]	2018	Selection	After	-	7
Jakubovski Filho et al. [41]	2019	Selection	Before	-	7
Bindewald et al. [29]	2019	Evaluation	During	Scores	7
Procter et al. [42]	2019	Selection	After	-	7
Le Calvar et al. [43]	2019	Selection	After	-	7
Zubcoff et al. [30]	2019	Modi cation, evaluation	After	Rankings	7
Alkhazi et al. [44]	2020	Selection	After	-	7
Alkhazi et al. [45]	2020	Selection	After	-	7
Silva et al. [46]	2020	Modi cation	Before	-	7
Arcega et al. [47]	2015	Selection	After	-	3
Font et al. [48]	2016	Selection	After	-	3
Font et al. [49]	2017	Selection	After	-	3
Marcén et al. [50]	2017	Selection	After	-	3
Ballar´n et al. [51]	2018	Selection	After	-	3
Pérez et al. [24]	2018	Selection	After	-	3
Pérez et al. [52]	2019	Selection	After	-	3
Pérez et al. [18]	2020	Selection	After	-	3
Our work	2020	Evaluation	During	HaFF _CM & HaFF _R	3

Marculescu et al. [28] allow domain specialists to change the relative importance of the objectives of a tness function for software testing. Bindewald et al. [29] incorporate decision-maker preferences (scores) into a multi-objective approach for product line architecture design. Zubcoff et al. [30] propose a pareto-based approach to assist requirement engineers to evaluate and prioritize requirements. Only Martinez et al. [25] completely replace the tness function with the human. Nevertheless, there are major differences between their work and our work. Their work does not provide evidence to support that the human is bene cial because their work does not compare the human with a non-human tness function. What their work does is to make a comparison between the human (combined with crossover and mutation operations) and Random Search. Their results show that the human improves the global score mean by only 0.5 points. Random Search achieves 4.20 and 3.95 in two experiments, while the human achieves 4.65 and 4.40 in the same experiments. The results do not seem to argue in favor of involving the human. In addition, their work is evaluated in the context of an academic case study. Conversely, our work shows that the combination of HaFF and the reformulation operation is bene cial at the industrial scale.

Furthermore, we completed the related work section with our previous works that deal with FLiM (see the eight works in the bottom part of the table). These works focus on the in uence of different factors on FLiM: run-time model traces [47], Formal Concept Analysis (FCA) [48], search strategies [49], learning to rank [50], synthetic problems [51], tness function [24], feature description collaboration [52],

and genetic operations [18]. None of these works leverage the use of the human as the tness function as our work does. Nevertheless, these works evaluate different tness functions to calculate the similarity between the feature description and each model fragment: Latent Semantic Indexing (LSI) [53], FCA, learning to rank, understandability [54], timing [55], and combinations of these [24]. Considering all of the works, LSI is the tness function that achieves the best results and, consequently, is the one that we have used in the baselines of this work.

3 Background

This section presents the main ideas of genetic operations and the model fragment population that are relevant for the baselines and HaFF. It also presents the LSI technique that the baselines use for the tness function. Finally, this section presents the Natural Language Processing that is applied for the reformulation operation and the tness function of the baselines.

3.1 Model Fragment Population

The top-left part of Fig. 2 shows an example of a model of the industrial partner. The model is speci ed using the Domain-Speci c Language (DSL) that formalizes the control and management of the trains. The DSL conforms to MOF (the OMG metalanguage for de ning modeling languages that is widely used in the modeling community). The DSL has expressiveness to generate the C++ code that controls and manages the trains of the industrial partner [2]. We present a simpli ed equipment-focused subset of the DSL (due to intellectual property rights concerns), and we do not show the terms that are included in the relationships for the sake of legibility.

Speci cally, the example in the gure presents a converter assistance scenario where a High Voltage Equipment device (Pantograph) collects energy from the overhead wires and sends it to its Voltage Converter (Converter). The converter then powers its assigned Consumer Equipment: the HVAC (air conditioning system). There is also a battery that powers the PA (public address system).

To encode model fragments as individuals, each element from the model is assigned to a position in a binary string (see gray numbers next to each element). Then, the binary string is used to indicate the presence or absence of that particular model element in the model fragment. For example, Model Fragment 0 (MF_0) can be seen in the bottom-left part of Fig. 2. The bits corresponding to the elements present in the model fragment are set to 1 (bits 1-5), while the bits corresponding to model elements that are not present in the model fragment are set to 0 (bits 6-11). The binary string is the most commonly used encoding for this type of problem [18], [49].

To create the initial population, different techniques may be applied, such as the creation of random model fragments or the use of seed model fragments as the starting point that are mutated until the initial population is completed. Using seed model fragments reduces the search effort needed to nd optimal solutions, as used in this work. The top-right part of Fig. 2 shows a model fragment population.

2. Learn more at https://youtu.be/Ypcl2evEQB8.

3.2 Genetic Operations

Below, we outline the most common operations used in the literature for FLiM [18]. The mask crossover operation takes two individuals as parents and combines them to generate two new individuals that inherit part of the genes from one of the parents and the rest from the other. The middle part of Fig. 2 shows the application of the mask crossover operation to MF_0 (in red) and MF_1 (in blue). To combine the individuals, a random mask is generated indicating which genes will be inherited from the rst parent and which genes will be inherited from the second parent (the random mask generated indicates that genes 1-4 and 7-10 will be selected from MF_0 and genes 5,6, and 11 will be selected from MF_1). MF_{10} shows one of the results of the combination; each of the genes is colored based on the parent that it is inherited from. To generate the second individual, the opposite mask will be used (like the one used to obtain MF_{11}).

Then, the random mutation operation is applied to the new individuals where each of the genes has a probability of mutating to the opposite state (i.e., a model element present in the individual is removed or a model element not present in the individual is added). In Fig. 2, the random mutation is applied to MF_{10}, and gene 7 is turned from 1 to 0 (MF_{12}).

The above genetic operations rely on randomness. In contrast, the replacement reformulation operation [18] is based on textual reformulation techniques [20] and brings semantic information to the genetic operations. The operation is based on the Domain Speci city (DS) of each term, which is calculated as $DS(t) = \frac{1}{|MF|} \sum_{mf \in MF} DS(t; mf)$. The $DS(t; mf)$ compares the relative frequency of each term present in the model fragment that is used as parent for the new individual with the relative frequency across the model fragments of the population. Then, the terms with the highest domain speci city value are selected, and the model fragment is evolved to include those terms (the bits associated to those terms are set to 1 and the rest of the bits that correspond to the old terms are set to 0). The middle-right part of Fig. 2 shows an example of application of the operation. In this case, MF_0 has been reformulated to include the terms 'batteri' [3], 'convert', and 'failur', resulting in the model fragment depicted in the bottom-right corner of Fig. 2. After the genetic operations are applied, the resulting model fragments are ready to be evaluated by the tness function.

3.3 Fitness Function of the Baselines

The baselines use LSI [53] to calculate the similarity between the feature description and each model fragment of the population. LSI has been used in software engineering tasks such as feature location in source code [15], and it has obtained the best results for feature location tasks. LSI is also the most common tness used for feature location in models [24], [48], [49], [51], [52]. LSI constructs vector representations of the feature description and the text that is present in the model fragments and builds a term-by-document co-occurrence matrix that formalizes the textual similarity between the feature description and each of the model fragments. The matrix is then decomposed

3. Batteri is the root of battery, see Section 3.4.

Fig. 2. Example of Model Fragment Encoding and Genetic Operations

following Singular Value Decomposition [56], which results in a vector representing the latent semantic of each individual. Then, the cosine similarity between the vector of the feature description and the vector of each individual from the population is calculated and used as the tness value of the individual. The cosine similarity will be a value ranging from -1 (no similarity) to 1 (the same vector, maximum similarity). The tness function is as follows: similarity $(i_1) = \cos(\ ) = \frac{A \ B}{\|A\| \|B\|}$ where i_1 is an individual, A is the vector representing the latent semantic of i_1, B is the vector representing the latent semantics of the feature description, and the angle formed by the vectors A and B is .

For instance, since the feature description shown in the top-right part of Fig. 1 shares more terms with MF_1 (from the top part of Fig. 2) than with MF_2, they get different similarity values (0.93 for MF_1 and 0.81 for MF_2).

3.4 Natural Language Processing

Both the reformulation operation and LSI bene t from the application of Natural Language Processing (NLP) to homogenize the inputs (the feature description and the model fragments), as is common practice when dealing with natural language [57]. We selected the following NLP techniques to homogenize the text of the inputs since they obtained the best results in a previous work [58].

First, the text is tokenized into words, and different tokenizers are applied based on the type of text being processed (e.g., white space, camelCase, or underscore). Second, the Parts-Of-Speech (POS) tagging technique is applied to identify the grammatical role of each word, allowing some categories that do not contain relevant information (e.g., prepositions) to be ltered out. Third, stemming techniques are applied to reduce the words to their root, enabling an easier comparison of terms from the same family. Fourth, the Domain Term Extraction and Stopword Removal techniques are applied to automatically lter terms in or out. It is worth mentioning that, in the context of HaFF, model fragments are shown to humans without this processing in order to favor comprehensibility.

For example, after applying NLP, the feature description shown in the top-right part of Fig. 1 will result in the following set of tokens: 'pass', 2x'current', 2x'convert', 'assign', 'peer', 'coverag', 'case', 'overload', 'failur', and ' rst'. The text has been tokenized using white space, prepositions and conjunctions have been removed as part of the POS tagging, 'equipment' has been removed as part of the stopwords removal, and the resulting tokens have been stemmed.

4 Evaluation

This section presents the evaluation of our work: the research questions that our work tackles, the evaluation process, the measures, baselines, and statistical analysis that we use to answer the research questions.

4.1 Research questions

We aim to answer the following research questions:

RQ_1: What is the performance in terms of the solution quality using the widespread crossover and mutation operations (randomly generating new candidate solutions) in the baseline and in HaFF?

RQ_2: What is the performance in terms of the solution quality using the reformulation operation (leveraging the latent semantics that models hold) in the baseline and in HaFF?

RQ_3: How much is the performance in uenced in terms of the solution quality using HaFF compared to the baselines?

4.2 Planning and execution

Fig. 3 presents an overview of the process that is planned to answer each research question. The upper part of the gure shows the data set from the industrial case study, which was provided by our industrial partner CAF. CAF uses software models to generate the rmware to control the trains that

have been manufactured over years. The middle part of the gure shows statistics of the case study. The product family is made up of 23 models of trains where each model includes, on average, more than 1200 model elements and about 15 properties that serve to differentiate among them.

Fig. 3. Overview of the evaluation process to answer each research question

The data set also includes 58 feature descriptions in natural language and 58 feature seeds provided by the industrial partner as well as the approved feature realization (i.e., the model fragment that corresponds to each feature) that 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 following inclusion and exclusion criteria were de ned during the creation of the oracle and the selection of features by the industrial partner. In order to include an approved feature realization in the oracle, it must be included in the product models provided for the evaluation. If a feature is randomly selected more than once, it must be excluded to avoid duplicates. The model fragments that correspond to each feature have between 6 and 19 model elements (an average of 13.29 and a median of 13.50).

Our previous works [52], [59] discussed the complexity of the problem for FLiM. Font et al. [59] stated that the search space can be huge when searching in model artifacts (magnitudes of around 10^{150} for models of 500 elements). In [52], Pérez et al. estimated that the time that a group of domain experts needs to manually locate 121 features in a industrial family of software models is 30.17 years. Since the product family in this work is made up of 23 models of trains where each model includes, on average, more than 1200 model elements and about 15 properties, an engineer needs 4.79 days to manually locate each feature (also assuming that only 1 second is needed to consider a property of a model element). Therefore, we use a Single-Objective Evolutionary Algorithm (SOEA) as a means of ef ciently exploring the huge search space. The objective of the algorithm is to nd the model fragment that best realizes the feature being located. The lower part of Fig. 3 shows the two baselines and the two HaFF variants for FLiM that are used to answer the research questions. All of the variants include initialization of the model fragment population from feature seed. The options for the genetic operations are mask crossover operation plus random mutation operation or the replacement reformulation operation. The options for the tness are LSI or completely replacing the tness function with the human.

To replace the tness function with the human, 29 software engineers were randomly selected from 42 software engineers who work for the industrial partner. The selected engineers have been developing software from 1 to 15 years (with a mean of 6.90 years) for an average of 4.41 hours per day.

To avoid the learning effect in the two HaFF variants, we chose a crossover design where the engineers are divided into two groups (G1 and G2) through randomization [60]. In HaFF_CM, G1 locates Features 1-28 and G2 locates Features 29-58. In HaFF_R, the group order is exchanged, so G2 locates Features 1-28 and G1 locates Features 29-58.

In each iteration of HaFF, the engineer acts as the tness function by evaluating each individual (i.e., model fragment) using a seven-point Likert scale: from 7 (the model fragment that best realizes the feature being located) to 1 (the worst model fragment). In this way, it is possible to quantify how much each model fragment realizes the feature being located at a speci c iteration. For instance, if there is a model fragment in the 7th iteration that contains many model elements that realize the feature being located but misses a few model elements, it will be evaluated with a 5. In contrast, if there is a model fragment in the 7th iteration that does not include any relevant model elements for the feature being located, it will be evaluated with a 1. We selected a seven-point scale rather than a broad rating to reduce the fatigue of the engineers, as recommended in [23]. Once the model fragments of a speci c iteration are evaluated by the engineer, each model fragment has a probability of being selected as parent for the new individuals that are created using the genetic operations. Those model fragments with high tness values will have higher probabilities. As an example, if the engineer provides similar high tness values to two model fragments (e.g., 5), they will have higher probabilities of being parents than other model fragments that are evaluated with lower tness values (e.g., 2).

With regard to the time that each engineer spends on the evaluation, we chose a slot of two hours since it is the median duration of software engineering experiments [61]. Hence, each engineer participates in the location of four features in total (two features that are randomly assigned [60]

per HaFF variant). This decision was made taking into account that the rating of an individual could last an average of 15 seconds. Thus, the time that each engineer spends on the evaluation of individuals to locate the four features is 100 minutes: 10 (individuals for subjective evaluation) x 10 (iterations) = 100 (ratings/feature) * 15 (seconds/rating) = 1500 (seconds/feature) * 4 (2 features to be located per HaFF variant) = 6000 seconds (100 minutes). The remaining 20 minutes are to provide a tutorial before the evaluation is started and to add a margin that ensures the completion of the evaluation. To verify the experiment design, we conducted a pilot study [62] with one engineer, who did not take part in the final evaluation. We verified that the evaluation had a correct parametrization.

Answering RQ $_1$: To assess the performance in terms of solution quality of the widespread crossover and mutation operations in the baseline and in HaFF, we use the variants of Column 2 (identified as Baseline_CM) and Column 3 (identified as HaFF_CM) of the table that is shown in the lower part of Fig. 3. To answer this question, we executed 1798 independent runs of the evolutionary algorithm: 58 (features) x 1 (Baseline_CM) x 30 repetitions (as suggested by Arcuri and Fraser [63]) + 58 (features) x 1 (HaFF_CM).

As a result of each independent run, a ranking of model fragments is obtained in descending order based on the fitness score. The first model fragment of the ranking (i.e., the model fragment with the highest fitness value) is compared against the oracle, which is considered to be the ground truth. Once the comparison is performed, a confusion matrix is calculated.

The confusion matrix is a table that is often calculated to describe the performance of a classification model on a set of data (the best solution) for which the true values are known (from the oracle). Since the solution is a model fragment in our case, the granularity is at the level of model elements. Hence, the presence or absence of each model element is considered as a classification. The confusion matrix holds the results of the comparison between the model fragment from the oracle and the model fragment from the solution using four categories. These are: 1) True Positive (TP), model elements that are present in the model fragments of both the solution and the oracle; 2) False Positive (FP), model elements that are present in the solution but absent in the oracle; 3) True Negative (TN), model elements that are absent in both the solution and the oracle; and 4) False Negative (FN), model elements that are absent in the solution but present in the oracle.

From the results in the four categories of the matrix, it is possible to extract a report with measurements that evaluate the performance in terms of solution quality. Specifically, we derive three performance measurements that are widely accepted in the software engineering research community [64], [65], [66]: recall, precision, and F-measure. Recall $\frac{TP}{TP+FN}$ measures the number of elements of the oracle that are correctly retrieved by the proposed solution. Precision $\frac{TP}{TP+FP}$ measures the number of elements from the solution that are correct according to the oracle. F-measure $2\ \frac{Precision\ \ Recall}{Precision\ +Recall}$ corresponds to the harmonic mean of precision and recall. Recall and precision values can range from 0% to 100%. A value of 100% in both precision and recall means that the solution and the oracle are the same.

Afterwards, all of the data resulting from the performance measures was analyzed using statistical methods following the guidelines in [67]. The goal of our statistical analysis is to provide formal and quantitative evidence that the difference in performance between Baseline_CM and HaFF_CM is significant and that there is an impact on the comparison metrics (i.e., the differences were not obtained by mere chance).

To do this, the results are compared through the run of a statistical test to assess whether there is enough empirical evidence to claim that there is a difference between Baseline_CM and HaFF_CM. The statistical test uses the value of a performance measure per feature in Baseline_CM and HaFF_CM. Since Baseline_CM is run 30 repetitions per feature, there are 30 values of the performance measure per feature. Hence, the mean value of the 30 repetitions is calculated to obtain a single performance measure value per feature in the baseline. Two hypotheses are defined: 1) the null hypothesis, H_0, is typically defined to state that there is no difference when Baseline_CM and HaFF_CM are compared; and 2) the alternative hypothesis, H_1, states that there is a difference. A statistical test aims to verify whether H_0 should be rejected. The statistical tests provide a probability value, p-value. The lower the p-value of a test, the more likely that the null hypothesis H_0 (defined to state that there is no difference between Baseline_CM and HaFF_CM) is false. It is accepted that a p-value under 0.05 is statistically significant [67], and so H_0 can be considered false.

The test to be used depends on the properties of the data. Since our data does not follow a normal distribution, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data; however, the Quade test is the most powerful when working with real data [68]. Moreover, the Quade test has shown better results than the others when the number of algorithms is low [69].

Answering RQ $_2$: To assess the performance in terms of solution quality of the reformulation operation in the baseline and in HaFF, we use the variants of Column 4 (identified as Baseline_R) and Column 5 (identified as HaFF_R) of the table that is shown in the lower part of Fig. 3. To answer this question, we executed 1798 independent runs of the evolutionary algorithm: 58 (features) x 1 (Baseline_R) x 30 repetitions (as suggested by Arcuri and Fraser [63]) + 58 (features) x 1 (HaFF_R).

As described for answering RQ$_1$, the model fragment with the highest fitness value that is obtained from each independent run is compared against the oracle to calculate the confusion matrix, which is used to extract the report of performance measures (recall, precision and F-measure) per feature in Baseline_R and HaFF_R. Then, the Quade test is run to provide formal and quantitative evidence to assess whether the difference in performance is significant.

Answering RQ $_3$: To assess how much the quality of the solution is influenced using HaFF compared to the baselines, the magnitude of the improvement should be assessed through effect size measures. For a non-parametric

effect size measure, we used two non-parametric effect size measures: Vargha and Delaney's $\hat{A}_{12}$ [70], [71] and Cliff's delta [72].

$\hat{A}_{12}$ measures the probability that running HaFF yields higher values than running the baseline. If HaFF and the baseline are equivalent, then $\hat{A}_{12}$ will be 0:5. For instance, $\hat{A}_{12} = 0:65$ between Baseline_CM and HaFF_CM means that Baseline_CM would obtain better results in 65% of the runs, whereas $\hat{A}_{12} = 0:25$ means that HaFF_CM would obtain better results in 75% of the runs.

Cliff's delta is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap with values ranging from -1 to 1. For example, when comparing Baseline_CM and HaFF_CM: a value of 0 means there is no difference; a value of 1 means that all samples in Baseline_CM are lower than all samples in HaFF_CM; a value of 1 means the opposite (all samples in Baseline_CM are higher than all samples in HaFF_CM). Moreover, threshold values were defined [73] for the interpretation of Cliff's delta effect size ($|d| < 0:147$! "negligible"; $|d| < 0:33$! "small"; $|d| < 0:474$! "medium", $|d| \geq 0:474$! "large"). We recorded an $\hat{A}_{12}$ and a Cliff's delta value for each pairwise comparison in recall, precision and F-measure between Baseline_CM and HaFF_CM, and between Baseline_R and HaFF_R.

4.3 Implementation details

For a fair comparison between each baseline and each HaFF variant, we chose the parameters shown in Table 2 to calibrate the evolutionary algorithm and the fitness. These parameters (such as population size and number of parents) correspond to those settings that are commonly used in the literature [18], [23], [24], [49], [74], [75].

TABLE 2
Parameter settings

	Parameter description	Value
Evolutionary algorithm		
Baselines	Size: Population Size	100
HaFF	Size: Population Size	10
Crossover and mutation	: Number of Parents	2
	: Number of offspring from parents	2
	$p_{crossover}$: Crossover probability	0.9
	$p_{mutation}$: Mutation probability	0.1
Replacement reformulation	Number of relevant documents	5
Fitness		
LSI	k: Number of dimensions	100
HaFF	Iterations	10
	Individuals for subjective evaluation	10

To determine the stop condition in the two baselines, we ran some prior tests to determine the convergence time. During the tests, we allocated 30 minutes as the fixed amount of wall clock time to locate one feature. According to the tests, the time needed to converge was below 60 seconds for locating each feature. Therefore, we established the stop condition in 80 seconds (adding a margin to ensure convergence as recommended in [18]), ensuring that the baselines run enough time to obtain the best solutions in our work. During this time, Baseline_CM does an average of 34985 iterations and Baseline_R does an average of 398 iterations. To determine the stop condition in the two HaFF variants, we allocated a fixed number of iterations (10) as recommended in [23] to avoid engineers' fatigue. Although the number of iterations in HaFF seem to be minimal compared to the baselines, the engineers will provide the most intelligent search possible, which might have an effect on the required number of iterations to locate a similar (or better) solution. The execution of the two baselines was performed using a Mac Pro computer with an Intel Xeon E5-2697 V2 processor (clock speeds 2.7 GHz and 12 cores) and 64 GB of RAM. The computer was running macOS Catalina (10.15.6) as the hosting Operative System. Since the engineers' evaluation time of the individuals is extremely long from the perspective of a computer [23], the execution of HaFF was performed using Lenovo E330 laptops, with an Intel Core i5-3210M processor (clock speeds 2.5GHz and 2 cores) with 16GB RAM and Windows 10 64-bit as the hosting Operative System.

The implementation of the evolutionary algorithm, which includes the genetic operations (mask crossover operation, random mutation operation, and the replacement reformulation operation) as well as the LSI fitness were downloaded from the repository of [18] that is publicly available [76]. This implementation uses the Eclipse Modeling Framework [77] to manipulate the models and the Java(TM) SE Runtime Environment (build 1.8.0_77). The NLP techniques are implemented using OpenNLP [78] for the POS-Tagger, and the English (Porter2) stemming algorithm [79] for the stemming algorithm (originally created using snowball and then compiled to Java). Since the software models of the data set belong to trains that are currently operating and under maintenance contracts or will be released in the near future, this information is limited by confidentiality agreements with the industrial partner. Nevertheless, a screenshot of the industrial partner's modeling tool is available. The screenshot shows the evaluation of model fragments during an iteration of HaFF Moreover, for purposes of replicability, the csv files used as input in the statistical analysis as well as the R scripts that were implemented to analyze the results are available at: https://bitbucket.org/svitusj/haff-vs-baselines- -sbmde.

5 Results

5.1 Research Question 1

Fig. 4 shows the box plots with the distribution of the results of the performance values (in terms of recall, precision, and F-measure) that are obtained in both Baseline_CM and HaFF_CM, which use the widespread crossover and mutation operations. In addition, the upper part of Table 3 shows the mean values and standard deviations of the obtained results. In terms of recall, precision, and F-measure, Baseline_CM outperforms HaFF_CM, providing an average value of 56.35%, 52.29% and 51.68%, respectively.

The lower part of Table 3 shows the p-values of the Quade test that are obtained as a result of comparing Baseline_CM with HaFF_CM. Since the p-values are smaller than the 0.05 statistical significance threshold for all performance indicators, the hypothesis H_0 can be considered

Fig. 4. Performance values for Baseline_CM and HaFF_CM

false. Hence, there are signi cant differences between Baseline_CM and HaFF_CM for recall, precision and F-measure.

TABLE 3
Results using the widespread genetic operations

Performance measures	Baseline_CM	HaFF_CM
Recall ( )	56.35 13.76	26.96 14.14
Precision ( )	52.29 16.50	32.47 17.33
F-measure ( )	51.68 10.91	25.22 11.97
Statistical signi cance (p-value)		
Recall: $2{:}2\text{x}10^{-16}$	Precision: $3{:}573\text{x}10^{-7}$	F-measure: $2{:}2\text{x}10^{-16}$

RQ_1 answer. The results reveal that Baseline_CM (56.35% in recall, 52.29% in precision, and 51.68% in F-measure) outperforms HaFF_CM in all performance measures. Moreover, we can state that there are signi cant differences between Baseline_CM and HaFF_CM for all performance measures.

5.2 Research Question 2

Fig. 5 shows the box plots with the performance results in terms of recall, precision and F-measure of Baseline_R and HaFF_R, which use the reformulation operation. The upper part of Table 4 shows the mean values and standard deviations of recall, precision and F-measure. In all performance measures, HaFF_R outperforms Baseline_R. HaFF_R obtains an average value of 96.01% in recall, 82.34% in precision, and 88.34% in F-measure.

Fig. 5. Performance values for Baseline_R and HaFF_R

The lower part of Table 4 shows the p-values of the Quade test that are obtained as a result of comparing Baseline_R with HaFF_R. The p-valueis greater than the 0.05 statistical signi cance threshold for recall, whereas the p-values for precision and F-measure are lower than 0.05. Therefore, the hypothesis H_0 can be considered true for recall, so there is no difference between Baseline_R and HaFF_R in recall. In contrast, there are signi cant differences between Baseline_R and HaFF_R for precision and F-measure.

TABLE 4
Results using the reformulation operation

Performance measures	Baseline_R	HaFF_R
Recall ( )	95.86 2.38	96.01 3.30
Precision ( )	68.08 11.92	82.34 11.39
F-measure ( )	79.03 8.30	88.34 7.80
Statistical signi cance (p-value)		
Recall: 0.2719	Precision $9{:}478\text{x}10^{-10}$	F-measure: $2{:}595\text{x}10^{-9}$

RQ_2 answer. The results reveal that HaFF_R (96.01% in recall, 82.34% in precision and 88.34% in F-measure) outperforms Baseline_R in all performance measures. According to the classi cation provided by Hayes et al. [80], which classi es the results as acceptable, good, or excellent depending on recall and precision values, the results of HaFF_R are excellent for all 58 features (since all recall values are greater than 80% and all precision values are greater than 50%). In addition, we can state that there are signi cant differences between Baseline_R and HaFF_R in precision and F-measure.

5.3 Research Question 3

Table 5 shows the effect size statistics ($\hat{A}_{12}$ and Cliff's delta) between pair-wise comparisons of a baseline and HaFF. The upper part of the table shows the values obtained between Baseline_CM and HaFF_CM, whereas the lower part shows the values obtained between Baseline_R and HaFF_R.

Between Baseline_CM and HaFF_CM, the $\hat{A}_{12}$ value shows that Baseline_CM obtains better results than HaFF_CM in 92.84% of the runs for recall, in 79.71% of the runs for precision, and in 94.87% of the runs for F-measure. The magnitude of improvement using Baseline _CM instead of HaFF_CM can be interpreted as being large according to the magnitude scales [73] of the Cliff's Delta values.

TABLE 5
Effect size measures for comparing the baselines with HaFF

	Recall	Precision	F-measure
	Baseline_CM vs HaFF_CM		
$\hat{A}_{12}$	0.9284	0.7971	0.9487
Cliff's Delta	0.8567 (large)	0.5942 (large)	0.8974 (large)
	Baseline_R vs HaFF_R		
$\hat{A}_{12}$	0.4307	0.1938	0.2033
Cliff's Delta	-0.1385 (negligible)	-0.6124 (large)	-0.5933 (large)

Between Baseline_R and HaFF_R, the $\hat{A}_{12}$ values are 0.4307 for recall, 0.1938 for precision, and 0.2033 for F-measure, implying that HaFF _R obtains better results than Baseline_R in 56.93% of the runs for recall, 80.62% of the runs for precision, and 79.67% of the runs for F-measure. The magnitude of improvement using HaFF _R instead of Baseline_R can be interpreted as being negligible for recall and being large for precision and F-measure according to the magnitude scales [73] of the Cliff's Delta values.

RQ_3 answer. From the results, we can conclude how much the performance in terms of solution quality is influenced using HaFF compared to the baseline when the genetic operations are either the widespread crossover and mutation operations or the reformulation operation. Using the widespread genetic operations, the magnitude of improvement using Baseline_CM instead of HaFF_CM can be interpreted as being large in F-measure according to the magnitude scales [73] of the Cliff's Delta values. In contrast, using the reformulation operation, the magnitude of improvement using HaFF_R instead of Baseline_R can be interpreted as being large in F-measure according to the Cliff's Delta magnitude scales [73].

6 Discussion

We analyzed the results in order to understand what humans bring to the table that leads HaFF_R to better precision results. First of all, humans are not sensitive to vocabulary mismatch, while baselines' fitness are. Vocabulary mismatch occurs when different terms are used to refer to the same concept. NLP unifies terms using a predefined dictionary, but in-house cases remain an issue. For example, both PLC and COSMOS refer to an on-board programmable controller. PLC is the generic term that is widely used to refer to this kind of controller, whereas COSMOS is an in-house term that is used exclusively by the industrial partner for historical reasons (COSMOS was the codename of its first PLC). Another example is circuit breaker and HSBC. Both terms refer to an on-off switch. Circuit breaker is a generic term, whereas HSBC is the name of a popular brand of circuit breakers. The brand name became the class name for the industrial partner.

Baselines fail to establish the similarity between feature descriptions and model fragments when cases such as those above come into play. Nevertheless, we did not identify a single case where humans fail because of vocabulary mismatch. For example, in the best evaluated solutions by human, both PLC and COSMOS appear. In the solutions of the baselines, only PLC appears because the baselines cannot establish that COSMOS is the in-house alias that was informally given to PLC. When we asked humans about the above examples, they acknowledge that they identify them as synonyms without even thinking. Actually, they did not recall that these examples were documented anywhere since they are self-evident for anyone that works at the company.

Tacit knowledge is another factor where humans excel in comparison to baselines' fitness. Tacit knowledge is the knowledge that is difficult to transmit by means of writing it down or verbalizing it [81]. When humans produce feature descriptions, they unconsciously omit part of the description which is implicit for them. For example, humans use the term doors in feature descriptions, but they do not clarify the number of doors, whether the doors belong to one side or another of the train, or if the doors refer to cabin doors, car doors, or both. When we asked humans about this behavior, they argue that they do not omit information intentionally. Solutions by the baseline include all of the model elements that are related to doors. In contrast, the best evaluated solutions by human only include model elements that are related to the necessary doors.

Tacit knowledge hampers the performance of baselines. Baselines completely rely on the input feature descriptions. Incomplete feature descriptions prevent baselines from achieving better results. Conversely, the performance of humans does not suffer from tacit knowledge. For example, when humans read, as part of a feature description, that all doors have to open at every station, they claim that it is obvious that doors refer to car doors (not cabin doors) on the side of the station (not on the side of the tracks).

We also identified cases where HaFF_R is not recommendable. One might think that since the human is immune to vocabulary mismatch and tacit knowledge (because of their experience at the company), the human will always beat the baselines' fitness, but this is not the case. Humans argue that there are features that they are not familiar with despite the length of their experience time at the company. This was argued even by humans that reported the highest experience time values. For example, there are humans who have the maximum experience time working with the features of the braking system of the train (15 years), but they are not familiar with the features of the consumer equipment of the trains. These experienced humans obtain worse results locating features related to the consumer equipment of the trains (such as the air conditioning) than other humans who have less experience time (2 years) but who are familiar with those features.

We delved deeper in order to better understand this phenomenon, and we asked them to rate their familiarity for each feature by means of a Likert scale ranging from 7 (the highest self-rated familiarity) to 1 (the lowest self-rated familiarity). The self-rated familiarity is supplied by the human before the feature is located in the software models. Hence, the self-rated familiarity relies on the human's experience in the development or maintenance of the feature being located. It turns out that when the human is not familiar with the particular feature to be located (values from 1 to 3), the HaFF_R results are worse than the results of the best baseline for the same feature.

In light of the above, we recalculated the performance indicators of HaFF_R considering only those cases where the feature was rated with a familiarity value greater than 3. HaFF_R increased the difference to the best baseline from 0.15% to 1.15% in recall, and from 14.26% to 20.05% in precision. Therefore, to better leverage HaFF_R, it is important to correctly identify when each human should use HaFF_R or Baseline_R for each case.

After delving deeper, we learned the following: 1) what made the human stronger than the baselines; 2) taking into account self-rated familiarity instead of experience time increased the difference to the best baseline; 3) the baselines still hold (good results) when the human is not familiar with the feature being located; and 4) how to use self-rated familiarity in order to decide when to use HaFF_R or Baseline_R. Self-rated familiarity is not optimal, but it is near-optimal. We found only two cases where HaFF_R with familiarity values greater than 3 performed worse than the best baselines. Further research is still required to evaluate the self-rated familiarity, the scale, and the threshold in other domains and tasks. Identifying when a HaFF or non-HaFF approach should be used is relevant because in industrial domains where software has been developed for decades,

such as the one of the industrial partner, no single human knows all of the features of the company.

We also analyzed the results to understand why the reformulation operation made HaFF_R beneficial, while crossover and mutation (HaFF_CM) failed. The reformulation operation produced significantly better candidate solutions than crossover and mutation. After all, crossover and mutation rely on randomness to produce new solutions while the reformulation operation relies on domain specificity. Although the random operations of HaFF_CM use the model fragments that engineers rated with high fitness values as parents, the operations randomly add/remove elements to produce new model fragments. In contrast, the idea of domain specificity of the reformulation operation of HaFF_R yields better quality results because the elements that are added/removed to produce new model fragments take into account the domain specificity of the terms of the model fragments, which engineers rated with high fitness values. What we know is that, in the case of FLiM, HaFF requires smart operations that offer the human better solutions than the ones produced by the random nature of crossover and mutation. What we do not know is if the reformulation operation can enable HaFF in the context of other software engineering tasks, or if other tasks require new smart operations.

Furthermore, we ran a focus group to acquire feedback from the 29 software engineers from our industrial partner who were involved in HaFF acting as the fitness function. Specifically, the focus group was composed of the following open questions: (1) What do you think about the results of each approach (HaFF and baselines)? (2) How did you feel locating features in models with HaFF? and (3) How do you imagine the use of HaFF for more complex problems?

The engineers stated that the results of HaFF were superior to the results of the baselines. They acknowledge that the solutions of the baselines included mistakes that were easily avoided when they guided the search themselves. The engineers described the phenomena of vocabulary mismatch and tacit knowledge using their own words.

Another aspect that the engineers highlighted as an advantage of HaFF over the baselines is that HaFF allows them to be involved during the search process without having to manually explore the entire software models. A few of the engineers referred to the “cyberpart” of HaFF as a sherpa. Engineers not only acknowledged this as being relevant to the daily work maintaining features in models, but they also found that the process of evaluating the solutions was fast and easy using a predefined scale. The engineers mentioned that they did not find the process long or exhausting during the 10 iterations that should be performed per feature. In fact, they found it interesting to see how the solutions were improved iteration after iteration.

The engineers also mentioned that HaFF can be used for more complex problems. To do this, they imagined that HaFF would require more iterations to find good solutions, and they proposed that more than one engineer participate in the evaluation of solutions of the feature being located. Thus, more iterations could be done without increasing the required time per engineer.

7 A GUIDELINE FOR APPLYING HAFF TO OTHER SOFTWARE ENGINEERING PROBLEMS

In the case of reformulating a software engineering problem into a search problem (SBSE), there are three main ingredients (encoding, operations, and fitness), with fitness being the most important one. In fact, many SBSE works resort to binary encoding and use widespread crossover and mutation operations (as is the case with most SBMDE works [18]). However, fitness may require a great effort in order to create the one that gives the desired results (e.g., it has taken us many years to explore a multitude of techniques for the fitness function in the problem of feature location in models). Using the Human as the Fitness Function (HaFF) may be an alternative for creating the fitness, and HaFF may even perform better than the best fitnesses created to date for the problem at hand. To help researchers who want to explore HaFF in other software engineering problems, we propose the following guideline.

The first step (HaFF-preparation) is to rethink the operations in order to reach a number of required iterations that is compatible with using the Human as the Fitness Function (i.e., Takagi's recommendations [23]). This step can be achieved before using the human as fitness function. Instead, this step can be achieved by using an existing classic fitness or an optimal fitness (presented below). This is described as follows:

- P.1) Rethinking the operations. Many SBSE works use operations that rely on randomness to produce new candidate solutions. These operations may require a number of iterations so large that it is impossible for the human to evaluate all of the solutions. In HaFF, the operations become the most important ingredient. The success of HaFF may depend on the operations. This means that we must consider operations beyond the widespread crossover or mutation (e.g., looking at the operations catalog of this recent survey [82]), or we must even design new operations that explore the search space more intelligently (e.g., taking as inspiration how new model fragment reformulation operations were designed [18]).
- P.2) Bootstrapping HaFF. In some software engineering problems, there may be a previous SBSE approach, as occurs in the case of feature location in models. That approach can be used as a starting point to rethink the operations for HaFF. However, applying HaFF to a new problem does not always require a classic SBSE approach (non-HaFF) beforehand. One may think that a classic fitness is at least necessary. However, creating the classic fitness can be difficult. There may be an alternative for those problems where it is possible to use (or create) an oracle. An oracle (or golden set) is a set of problems and the solutions to those problems. If there is an oracle, it may be possible to design an inexpensive optimal

tness that is similar to the following equation:

$$fitness(m) = \sum_{i=0}^{n} \frac{g(i)}{n}$$

$$g(i) = \begin{cases} 1 & \text{if } value(m;i) = value(o;i) \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

where: m = any given individual
n = size of the individual
o = solution from the oracle.

The tness is the result of adding the g(i) values for all of the genes present in the individual (from 0 to n) and dividing the sum by the size of the individual. The g(i) is 1 if the gene value is the same in the individual and in the oracle (value(m, i) = value(o, i)) or -1, otherwise. Our previous work [59] shows this optimal tness applied to the feature location problem in models; however, it is possible to adapt the idea to other problems. It is important to note that this optimal tness is useful only for rethinking the operations, but it cannot be applied in the wild since not all solutions are known for all problems.

P.3) When are the operations good enough for HaFF? Following Takagi's recommendations to avoid human fatigue, solutions should be found for population sizes and number of iterations of 10 or 20. When using a classic tness, it should be taken into account that the human will provide the most intelligent search possible. Therefore, even though the combination of the operations and the classic tness requires a larger number of iterations, the nal combination of operations with the human may require a smaller number of iterations that is compatible with Takagi's recommendations. This was the case of our experiments. When using an optimal tness to rethink the operations for HaFF, we recommend following the recommendations as much as possible. Despite his/her intelligence, the human will not provide better solutions than the optimal tness.

The second step (HaFF-execution) is where the actual human plays the role of the tness function. Human fatigue might arise in this step. Points E.3 and E.4 can avoid human fatigue, but, in some cases, we should go back to the rst step to further rethink the operations. This is described as follows:

E.1) Who should play the role of HaFF? Intuition tells us that years of experience can make a difference in HaFF. It seems like a safe bet that the humans who have more experience with the problem at hand are the ones who should play the role of HaFF. Our work shows that the length of experience time is not always the best criterion. Humans who have more experience time but are unfamiliar with parts of the system guide the search worse than humans who have less experience time but are more familiar with. In our case, self-rated familiarity leads to better results than experience time. For this reason, we discourage using experience time as a criterion for choosing humans by default. Other criteria than experience time should be considered (e.g., self-rated familiarity), or other criterion speci c to the problem should be explored.

E.2) How should humans evaluate the solutions? HaFF is underexplored in software engineering problems. Our work shows that the use of a Likert scale has been successful in a problem like ours. Our recommendation is to use the scale as a starting point and adjust it to a coarser or ner grain technique depending on the needs of the problem.

E.3) Human collaboration in HaFF. Even though humans do not collaborate with each other in our experiments, it may make sense in other software engineering problems for several humans to collaborate in HaFF in order to share the burden of evaluating the solutions. This collaboration could lead to a higher number of iterations or a higher population size. In other words, for some kind of problems a single human may fail at HaFF, but the collaboration of a set of humans could led to success for the same problems.

E.4) Hybridizing HaFF. For problems in which fatigue appears, a hybrid approach could be explored in which a classical tness is executed a number of iterations (e.g., 1000) between each evaluation performed by the human. In our case, the human guides the search better than the baseline because the human is immune to the baseline problems (vocabulary mismatch and tacit knowledge). Our solution is pure HaFF because we did not reach human fatigue in our experiments.

The third step (HaFF-simulation) covers the case where HaFF has produced so much data that the human could be replaced by a machine-learning counterpart. This is described as follows:

S.1) Leveraging HaFF to achieve a Simulated Human as a Fitness Function (SHaFF). There are approaches that use machine learning to simulate the human and use that simulation in the tness function. Depending on the problem, simulating the human with machine learning may require many examples for training that are not available from the beginning. Addressing a problem with HaFF can help to generate the examples as HaFF is used. In some problems, HaFF can be used until enough examples are available to train a simulation of the human for that problem.

Alternatively, HaFF can also play the role of a sanity check (HaFF-sanity check) when a sophisticated classic tness is created. This is described as follows:

SC.1) HaFF as a form of sanity-check. The search-based community might consider the use of HaFF to check that sophisticated tness functions perform better than a humble human guiding the search within Takagi's recommendations. Since a classic search approach is designed without HaFF in mind, the

classic tness is expected to outperform the human. This could be tested by making a comparison of the search approach with a variant of the search approach where the human replaces the classic t-ness, and the variant is executed within Takagi's recommendations.

To conclude, not all software engineering problems are equal, which can be re ected in the effort that it may take to evaluate a solution by a human. In the case of model-based software development problems, the effort should be less than in software development problems using source code (or the extreme case of binary code). Given the initial phase in which the research of HaFF applied to software problems is currently, it seems reasonable to continue its exploration in other problems with a high level of abstraction where less human effort is required to evaluate solutions. Next, we provide a few examples that might help give a more precise picture on the detection of opportunities for using HaFF.

In the context of Search-based Bug Location in the models of induction hobs from BSH (including Bosch, Siemens, and Gaggenau brands among others), we highlighted how the interaction of humans in the evaluation of solutions could in uence the quality of the results [83]. In this domain, the software that controls the induction hobs is responsible for recon guring them in the event of context changes. As an example, the software turns inductors on or off in response to pot movements on the surface of induction plates. On the one hand, research showed that humans were bene cial when evaluating model fragments that highlighted elements of the model that could be relevant for the location of a bug. On the other hand, human interaction was detrimental to the process when evaluating recon guration strings that could lead to the bug. In both cases, the humans evaluated only the results of the ranking produced by the search.

Having identi ed how humans are bene cial in evaluating solutions, one might wonder if there is an opportunity to use HaFF. It is true that the number of results evaluated during the search was much higher than Takagi's recommendations to avoid fatigue, however, the operations used to explore the search space were based on the randomness of crossover and mutation, which suggests that there is room to reduce the number of results to evaluate. The combination of 1) we know how to present the solutions to humans to be bene cial, and 2) there are too many solutions because of random operations might be a pattern to detect the opportunity to use HaFF (or hybrid HaFF) using the guideline of this section.

In the context of Procedural Content Generation in the Kromaia video game (PlayStation 4 and Steam), we show how a search-based approach could generate Non-Playable Characters (NPCs), speci cally the nal enemies that appear at the end of each phase of the video game [84]. This work did not address the artistic part of the NPCs, rather, it addressed the part of the software models that describes their architecture and behavior. The part of software engineering that addresses game development is known as Game Software Engineering (GSE) [85].

GSE can be an opportunity to use HaFF. The main enemy of HaFF is the fatigue of humans when evaluating solutions. However, in GSE humans may consider playing video games as entertainment, rather than a burden. It is true that the perception of entertainment may depend on factors such as the diversity or dif culty of the content, but GSE has the potential to use HaFF. Furthermore, it is not uncommon for a blockbuster video game to be played by millions of players, which may also be an opportunity to recruit enough players to achieve human collaboration in HaFF (see E.3 of the guideline).

8 Threats to validity

To acknowledge the threats to the validity of our work, we use the classi cation suggested by De Oliveira et al. [86].

1) Conclusion Validity threats. To address the lack of statistical tests, we employed standard statistical analysis following accepted guidelines [63]. The not accounting for random variation threat was addressed by considering 30 independent runs for each feature description in the baselines. In HaFF, it is not possible to consider 30 independent runs per feature since the human's availability is a limited resource and the results of additional runs for the same feature would be in uenced by the learning effect. For the lack of a good descriptive analysis threat, we have analyzed the confusion matrix obtained using recall, precision, and F-measure measurements; however, other measurements could be applied. In addition, some works argue that the use of the Vargha and Delaney $\hat{A}_{12}$ measurement can be misrepresentative [63] and that the data should be pre-transformed before applying it. We did not nd any use cases for data pre-transformation that applied to our case study. Nevertheless, we also used Cliff's delta as an effect size measure.

2) Internal Validity threats. We mitigated the learning effect threat using a crossover design for the evaluation. In addition, the engineers who participated in the evaluation were not involved in the oracle provided by the industrial partner. With regard to the information exchange threat, the engineers were not able to exchange information during the evaluation. The understandability threat was mitigated by providing a tutorial before the evaluation started. The fatigue effect was mitigated by: 1) setting the number of iterations and the number of individuals of the algorithm per feature to 10 as recommended in the literature [23]; and 2) establishing a total time of two hours for the whole evaluation (including the tutorial) since this time is the median duration of software engineering experiments [61]. We mitigated the researcher bias threat [87] by not participating in the selection of the engineers. The imbalanced group of subjects threat was mitigated by randomly assigning each engineer to a group and to the features to be located. With regard to the poor parameter settings threat, we mitigated it by using values that have been tested in similar algorithms for FLiM [18]. Default values are good enough to measure the performance of location techniques as suggested by Arcuri and Fraser [63]. To address the threat of the lack of real problem instances, the evaluation of this work was applied to an industrial case study.

3) Construct Validity threats. To address the lack of assessing the validity of cost measures threat, we performed our evaluation using three measures (recall, precision, and

F-measure), which are widely accepted in the software engineering research community [64]. Moreover, we performed a fair comparison between the baselines and HaFF.

4) External Validity threats. To mitigate the threat of the lack of a clear object selection strategy, the evaluation uses an industrial case study, whose instances are collected from real-world problems. To mitigate the generalizationthreat, HaFF is generic and is not only applicable to feature location and the domain of the industrial partner, but it is also applicable to other domains and to other software engineering tasks. Nevertheless, HaFF should be applied to other domains and tasks before assuring its generalization.

9 Conclusion

In this work, we study the in uence of using the Human as the Fitness Function (HaFF) on the quality of the results. To do this, we focus on SBMDE since models (the fundamental artifacts of SBMDE) should be a more favorable scenario than code for HaFF because of their higher level of abstraction. Furthermore, Pérez et al. [18] recently reported that, in the context of feature location in models, reformulation operations signi cantly reduce the required iterations in comparison to the widespread crossover and mutation operations. Our results show that HaFF leads to signi cantly better precision than the best baseline because humans are immune to the main limitations of the baselines: vocabulary mismatch and tacit knowledge. HaFF achieved better results only when HaFF was combined with the replacement reformulation operation (HaFF_R). We learned that a self-rated feature familiarity assessment helps to boost the quality of the results. In addition, a focus group interview con rms the engineers' acceptance of HaFF. Our results can motivate other researchers to evaluate HaFF in more software engineering tasks and other software artifacts. We provide a guideline that further discusses how to apply HaFF to other software engineering problems and provides recommendations for types of problems that should be used to continue exploring HaFF. For future work, we are planning to explore the potential bene ts of HaFF for bug location.

Acknowledgments

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

References

[1] M. Harman and B. F. Jones, "Search-based software engineering," Inf. Softw. Technol., vol. 43, no. 14, pp. 833–839, 2001.

[2] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in 8th IEEE International Conference on Software Testing, Veri cation and Validation, ICST 2015, Graz, Austria, April 13-17, 2015. IEEE Computer Society, 2015, pp. 1–12.

[3] A. Ramírez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," IEEE Transactions on Software Engineering, vol. 45, no. 8, pp. 760–781, 2019.

[4] A. A. Araújo, M. Paixao, I. Yeltsin, A. Dantas, and J. Souza, "An architecture based on interactive optimization and machine learning applied to the next release problem," Automated Software Engg., vol. 24, no. 3, p. 623–671, Sep. 2017.

[5] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," Inf. Softw. Technol., vol. 55, no. 1, p. 173–187, Jan. 2013.

[6] A. Ghannem, G. El Boussaidi, and M. Kessentini, "Model refactoring using interactive genetic algorithm," in Search Based Software Engineering, G. Ruhe and Y. Zhang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 96–110.

[7] C. L. Simons, J. Smith, and P. White, "Interactive ant colony optimization (iaco) for early lifecycle software design," Swarm Intelligence, vol. 8, no. 2, pp. 139–157, June 2014.

[8] L. Troiano, C. Birtolo, and R. Armenise, "A validation study regarding a generative approach in choosing appropriate colors for impaired users," SpringerPlus, vol. 5, no. 1, p. 1090, July 2016.

[9] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L. B. Said, "On the use of machine learning and search-based software engineering for ill-de ned fitness function: A case study on software refactoring," in Search-Based Software Engineering, C. Le Goues and S. Yoo, Eds. Cham: Springer International Publishing, 2014, pp. 31–45.

[10] C. Birtolo, P. Pagano, and L. Troiano, "Evolving colors in user interfaces by interactive genetic algorithm," in 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC). IEEE, 2009, pp. 349–355.

[11] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," Autom. Softw. Eng., vol. 24, no. 2, pp. 233–294, 2017.

[12] M. Kessentini, P. Langer, and M. Wimmer, "Searching models, modeling search: On the synergies of SBSE and MDE," in 2013 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), May 2013, pp. 51–54.

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

[14] J. Krüger, T. Berger, and T. Leich, "Features and how to nd them: A survey of manual feature location," in Software Engineering for Variability Intensive Systems - Foundations and Applications. Taylor & Francis Group, 2019, pp. 153–172.

[15] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," Journal of Software: Evolution and Process, vol. 25, no. 1, pp. 53–95, 2013.

[16] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 420–432, Jun. 2007.

[17] J. Wang, X. Peng, Z. Xing, and W. Zhao, "An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions," in Proceedings of the 27th Conference on Software Maintenance. IEEE, 2011, pp. 213–222.

[18] F. Pérez, T. Ziadi, and C. Cetina, "Utilizing automatic query reformulations as genetic operations to improve feature location in software models," IEEE Transactions on Software Engineering, June 2020. [Online]. Available: https://doi.org/10.1109/TSE.2020.3000520

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

[20] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 245–254.

[21] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in Proceedings of the 2013 International Conference on Software Engineering, ser. ICSE '13, 2013, pp. 842–851.

[22] M. M. Rahman and C. K. Roy, "STRICT: information retrieval based search term identi cation for concept location," CoRR, vol. abs/1807.04475, 2018.

[23] H. Takagi, "Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation," Proceedings of the IEEE, vol. 89, no. 9, pp. 1275–1296, September 2001.

[24] F. Pérez, R. Lapeña, J. Font, and C. Cetina, "Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study," Information & Software Technology, vol. 103, pp. 188–201, 2018.

[25] J. Martinez, J.-S. Sottet, A. G. Frey, T. Ziadi, T. Bissyandé, J. Vanderdonckt, J. Klein, and Y. Le Traon, Variability Management and

Assessment for User Interface Design. Cham: Springer International Publishing, 2017, pp. 81–106.

[26] J. Martinez, J.-S. Sottet, A. G. Frey, T. F. Bissyandé, T. Ziadi, J. Klein, P. Temple, M. Acher, and Y. le Traon, "Towards estimating and predicting user perception on software product variants," in New Opportunities for Software Reuse, R. Capilla, B. Gallina, and C. Cetina, Eds. Cham: Springer International Publishing, 2018, pp. 23–40.

[27] W. Kessentini, M. Wimmer, and H. Sahraoui, "Integrating the designer in-the-loop for metamodel/model co-evolution via interactive computational search," in Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, ser. MODELS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 101–111.

[28] B. Marculescu, R. Feldt, R. Torkar, and S. Poulding, "Transferring interactive search-based software testing to industry," Journal of Systems and Software, vol. 142, pp. 156 – 170, 2018.

[29] C. V. Bindewald, W. M. Freire, A. M. M. M. Amaral, and T. E. Colanzi, "Towards the support of user preferences in search-based product line architecture design: An exploratory study," in Proceedings of the XXXIII Brazilian Symposium on Software Engineering, ser. SBES 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 387–396.

[30] J. Zubcoff, I. Garrigós, S. Casteleyn, J.-N. Mazón, J.-A. Aguilar, and F. Gomariz-Castillo, "Evaluating different i*-based approaches for selecting functional requirements while balancing and optimizing non-functional requirements: A controlled experiment," Information and Software Technology, vol. 106, pp. 68 – 84, 2019.

[31] Y. Lin, X. Peng, Y. Cai, D. Dig, D. Zheng, and W. Zhao, "Interactive and guided architectural refactoring with search-based recommendation," in Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 535–546.

[32] T. Yue, S. Ali, H. Lu, and K. Nie, "Search-based decision ordering to facilitate product line engineering of cyber-physical system," in 2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2016, pp. 691–703.

[33] L. Van Rooijen and H. Hamann, "Requirements specification-by-example using a multi-objective evolutionary algorithm," in 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW), 2016, pp. 3–9.

[34] H. Lu, T. Yue, S. Ali, and L. Zhang, "Nonconformity resolving recommendations for product line configuration," in 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), 2016, pp. 57–68.

[35] C. Debreceni, I. Ráth, D. Varró, X. De Carlos, X. Mendialdua, and S. Trujillo, "Automated model merge by design space exploration," in Fundamental Approaches to Software Engineering, P. Stevens and A. Wasowski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 104–121.

[36] E. Batot and H. Sahraoui, "A generic framework for model-set selection for the unification of testing and learning mde tasks," in Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, ser. MODELS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 374–384.

[37] M. Fleck, J. Troya, and M. Wimmer, "Search-based model transformations," J. Softw. Evol. Process, vol. 28, no. 12, p. 1081–1117, Dec. 2016.

[38] P. Gómez-Abajo, E. Guerra, and J. de Lara, "A domain-specific language for model mutation and its application to the automated generation of exercises," Computer Languages, Systems & Structures, vol. 49, pp. 152 – 173, 2017.

[39] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Designing robust software systems through parametric markov chain synthesis," in 2017 IEEE International Conference on Software Architecture (ICSA), 2017, pp. 131–140.

[40] A. Kolchin, "Interactive method for cumulative analysis of software formal models behavior," PROBLEMS IN PROGRAMMING, no. 2-3, pp. 115–123, 2018.

[41] H. L. Jakubovski Filho, T. N. Ferreira, and S. R. Vergilio, "Preference based multi-objective algorithms applied to the variability testing of software product lines," Journal of Systems and Software, vol. 151, pp. 194 – 209, 2019.

[42] S. Procter and L. Wrage, "Guided architecture trade space exploration: Fusing model based engineering design by shopping," in 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 117–127.

[43] T. L. Calvar, F. Chhel, F. Jouault, and F. Saubion, "Toward a declarative language to generate explorable sets of models," ser. SAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1837–1844.

[44] B. Alkhazi, C. Abid, M. Kessentini, D. Leroy, and M. Wimmer, "Multi-criteria test cases selection for model transformations," Automated Software Engineering, pp. 1–28, 2020.

[45] B. Alkhazi, C. Abid, M. Kessentini, and M. Wimmer, "On the value of quality attributes for refactoring atl model transformations: A multi-objective approach," Information and Software Technology, vol. 120, p. 106243, 2020.

[46] D. N. A. d. Silva, "Adaptation oriented test data generation for adaptive systems," in 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), 2020, pp. 1–7.

[47] L. Arcega, J. Font, Ø. Haugen, and C. Cetina, "Leveraging models at run-time to retrieve information for feature location," in Proceedings of the 10th International Workshop on Models@run.time co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015), Ottawa, Canada, September 29, 2015, 2015, pp. 51–60.

[48] J. Font, L. Arcega, O. Haugen, and C. Cetina, "Feature location in models through a genetic algorithm driven by information retrieval techniques," in Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, ser. MODELS '16. ACM, 2016, pp. 272–282.

[49] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Achieving feature location in families of models through the use of search-based software engineering," IEEE Transactions on Evolutionary Computation, vol. 22, no. 3, pp. 363–377, 2018.

[50] A. C. Marcén, J. Font, O. Pastor, and C. Cetina, "Towards feature location in models through a learning to rank approach," in Proceedings of the 21st International Systems and Software Product Line Conference - Volume B, ser. SPLC '17, 2017, p. 57–64.

[51] M. Ballarín, A. C. Marcén, V. Pelechano, and C. Cetina, "Measures to report the location problem of model fragment location," in Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018, 2018, pp. 189–199.

[52] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," Automated Software Engineering, vol. 26, no. 1, pp. 161–202, 2019.

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

[54] H. Störrle, "On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters," in Proceedings of 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014), 2014, pp. 518–534.

[55] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining Version Histories to Guide Software Changes," in Proceedings of the 26th International Conference on Software Engineering, ser. ICSE '04, 2004, p. 563–572.

[56] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," Discourse processes, vol. 25, no. 2-3, pp. 259–284, 1998.

[57] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in Proceedings of the 2003 conference on Empirical methods in natural language processing, 2003, pp. 216–223.

[58] R. Lapeña, J. Font, O. Pastor, and C. Cetina, "Analyzing the impact of natural language processing over feature location in models," in GPCE 2017 - 16th International Conference on Generative Programming: Concepts & Experiences, 2017.

[59] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility," Software and Systems Modeling, 2021. [Online]. Available: https://doi.org/10.1007/s10270-021-00870-5

[60] H. J. Seltman, "Experimental design and analysis," Online at: http://www.stat.cmu.edu/ hseltman/309/Book/Book.pdf, 2012.

[61] D. I. K. Sjoeberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N. . Liborg, and A. C. Rekdal, "A survey of controlled experiments in software engineering," IEEE Transactions on Software Engineering, vol. 31, no. 9, pp. 733–753, 2005.

[62] M. Asadi, S. Soltani, D. Gasević, and M. Hatala, "The effects of

visualization and interaction techniques on feature model configuration," *Empirical Software Engineering*, pp. 1–38, 2014.
[63] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.
[64] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.
[65] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," in *Proceedings of the 11th Working Conference on Reverse Engineering*, ser. WCRE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 214–223.
[66] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 18–44, 2011.
[67] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, May 2014.
[68] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.
[69] W. Conover, *Practical nonparametric statistics*, 3rd ed., ser. Wiley series in probability and statistics. New York, NY [u.a.]: Wiley, 1999.
[70] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.
[71] R. J. Grissom and J. J. Kim, *"Effect sizes for research: A broad practical approach*. Mahwah, NJ: Earlbaum, 2005.
[72] N. Cliff, *Ordinal methods for behavioral data analysis.* Lawrence Erlbaum Associates, Inc, 1996.
[73] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.
[74] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 465–474.
[75] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *ACM Comput. Surv.*, vol. 44, no. 1, pp. 1:1–1:50, Jan. 2012.
[76] "Model fragment reformulation variants as genetic operations for feature location in models," https://bitbucket.org/svitusj/mfr, 2020.
[77] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.
[78] "Apache opennlp: Toolkit for the processing of natural language text," https://opennlp.apache.org/, 2019.
[79] "English (porter2) stemming algorithm," http://snowball.tartarus.org/algorithms/english/stemmer.html, 2019.
[80] J. H. Hayes, S. K. Sundaram, and A. Dekhtyar, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Transactions on Software Engineering*, vol. 32, pp. 4–19, 2006.
[81] H. Collins, *Tacit and explicit knowledge*. University of Chicago Press, 2010.
[82] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *International Journal of Machine Learning and Computing*, vol. 7, no. 1, pp. 9–12, 2017.
[83] L. Arcega, J. Font, Ø. Haugen, and C. Cetina, "Bug Localization in Model-based Systems in the Wild," *ACM Transactions on Software Engineering and Methodology*, 2021.
[84] D. Blasco, J. Font, M. Zamorano, and C. Cetina, "An evolutionary approach for generating software models: The case of kromaia in game software engineering," *J. Syst. Softw.*, vol. 171, p. 110804, 2021.
[85] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: A systematic review," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 888–901, 2010.
[86] M. de Oliveira Barros and A. C. D. Neto, "Threats to validity in search-based software engineering empirical studies," *RelaTe-DIA*, vol. 5, 01 2011.
[87] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey." in *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE'2010)*, 2010, pp. 374–379.

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

Jaime Font received the Ph.D. degree in computer science from the University of Oslo, Oslo, Norway. He is Assistant Professor with the SVIT Research Group, Universidad San Jorge, Zaragoza, Spain. His current research interests include reverse engineering, evolutionary computation, and variability modeling. He publishes his research results and participates in high-level international software engineering conferences and journals, such as IEEE Transactions on Evolutionary Computation (TEVC), and Software and System Modeling (SoSyM) journal.

Lorena Arcega is Assistant Professor with the SVIT Research Group, Universidad San Jorge, Zaragoza, Spain. She received the Ph.D. degree in computer science from the University of Oslo, Oslo, Norway. Her current research interests include models at runtime, software maintenance and evolution, and variability modeling. She publishes her research results and participates in high-level international software engineering conferences and journals, such as the International Conference on Model Driven Engineering Languages and Systems (MODELS), and Software and System Modeling (SoSyM) journal.

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