Evolutionary Algorithm for Bug Localization in the Reconfigurations of Models at Runtime

Lorena Arcega
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
Department of Informatics
University of Oslo
Oslo, Norway
larcega@usj.es

Jaime Font
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
Department of Informatics
University of Oslo
Oslo, Norway
jfont@usj.es

Carlos Cetina
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
ccetina@usj.es

ABSTRACT

Systems with models at runtime are becoming increasingly complex, and this is also accompanied by more software bugs. In this paper, we focus on bugs appearing as the result of dynamic reconfigurations of the system due to context changes. We materialize our approach for bug localization in reconfigurations as an evolutionary algorithm. We guide the evolutionary algorithm with a fitness function that measures the similarity to the description of the bug report. The result is a ranked list of reconfiguration sequences, which is intended to identify the reconfiguration rules that are relevant to the bug. We evaluated our approach in BSH and CAF, two real-world industrial case studies, measuring the results in terms of recall, precision, F-measure and Matthews Correlation Coefficient (MCC). In our evaluation, we compare our approach with two other approaches: a baseline that is the one used by our industrial partners for bug localization and a random search as sanity check. Our study shows that our approach, which takes advantage of the reconfigurations of models at runtime, outperforms the other two approaches. We also performed a statistical analysis to provide evidence of the significance of the results.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Search-based software engineering**;

KEYWORDS

Bug Localization, Models at Runtime

ACM Reference Format:
Lorena Arcega, Jaime Font, and Carlos Cetina. 2018. Evolutionary Algorithm for Bug Localization in the Reconfigurations of Models at Runtime. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239392

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MODELS '18, October 14–19, 2018, Copenhagen, Denmark
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239392

1 INTRODUCTION

Model-Driven Engineering (MDE) is being applied in an ever increasing manner to cope with the complexity of software systems by raising the level of abstraction [23]. Models at runtime [6] is defined as *a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective* [7]. A recent survey [36] has classified the objective of the use of models at runtime in the following categories: adaptation [27, 35], monitoring, simulation and prediction [12, 16, 24], abstraction and platform independence [38], consistency and conformance [2], policy checking and enforcement [34], and error handling [10].

Software is becoming increasingly complex, and systems with models at runtime are not an exception. Unfortunately, an increase in complexity is accompanied by an increase in the appearance of software bugs. Hence, software maintenance is becoming more and more important. Lehman et al. [19] pointed out that up to 80% of the lifetime of a system is spent on maintenance and evolution activities. Software maintainers spend from 50% up to almost 90% of their time trying to understand a program to make changes correctly.

In a system with models at runtime, the models experience reconfigurations at runtime due to context changes, being these reconfigurations a source of bugs. A recent Search-Based Software Engineering survey [40] reveals that none of the Bug Location approaches take in account the bugs caused by the reconfigurations of a models at runtime system.

Our work is focused on locating bugs that appear as the result of dynamic reconfigurations of the system due to context changes. In this paper, we present an approach for bug localization in the reconfigurations that occur in runtime models called EBRo. We materialize our approach for bug localization in reconfigurations through an evolutionary algorithm. The solutions provided by our approach are sequences of reconfigurations that, when followed, might lead to the model at runtime which contains the located bug.

The evolutionary algorithm is guided by a fitness function that considers the similarity to the description of the bug report. To measure the textual similarity, we start from an initial model at runtime to which we apply a sequence of reconfigurations, obtaining another model in which we evaluate whether the modified elements are similar with the description of the bug. As a result, software engineers obtain a ranked list of sequences of reconfigurations,

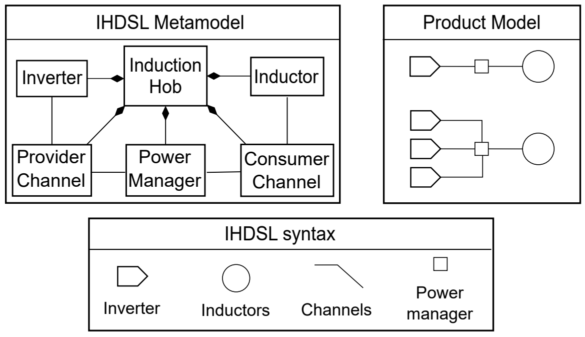


Figure 1: IHDSL metamodel, syntax, and product model

intended to identify the reconfiguration rules that are relevant to the bug.

We have applied our approach to the industrial domains of BSH and CAF. BSH is one of the larges manufacturers of home appliances in Europe. Its induction division has been producing induction hobs under the brands of Bosch and Siemens for the last 15 years. CAF produces a family of Programmable Logic Controller (PLC) software to manage the trains they manufacture, which has been under development for more than 25 years. The firmware from both industries is specified by means of Domain Specific Languages. The firmware of the products is generated from the DSL models and uses models at runtime to change the configuration when their products are in operation.

In our evaluation, we compare our approach with a baseline approach. Since there are no specific baselines for bug localization in reconfigurations of models at runtime, we use as a baseline the approach used by BSH and CAF for bug localization. In addition, we compare our approach with a random search approach (RS) as sanity check. We apply our EBRo approach, the baseline approach and the RS approach to the product families of BSH and CAF. They provided us with documentation about bugs. For each bug, the documentation provided a bug description, the reconfigurations that trigger the bug and the localization of the bug. Taking the bug descriptions, the set of reconfigurations, and an initial model of the product family as input, and the reconfigurations that trigger the bug and the location of the bugs as oracle (ground truth), we measure the results in terms of the standard measurements accepted by the software engineering community: recall, precision, F-measure (the combination of both recall and precision), and Matthews Correlation Coefficient (MCC) [25, 33].

EBRo performed better than the other algorithms in terms of the four measured performance indicators. On average, up to 78.72% of the reconfigurations that were expected to trigger the bugs being located (according to the oracle) were found when EBRo was used (up to 66.84% for the baseline, and 38.42% for the random search approach). It turns out that the genetic operations performed by the EBRo approach with the fitness function are able to properly traverse search spaces originated when locating bugs over runtime reconfigurations of the system.

The remainder of the paper is structured as follows. In Section 2, we present the Domain Specific Language used by one of our industrial partner. In Section 3, we explain the motivation for bug

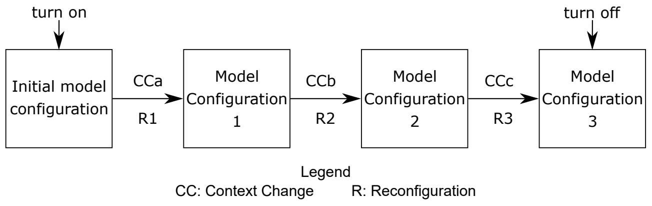


Figure 2: Induction Hob at runtime

localization in reconfiguration rules. In Section 4, we describe our bug localization approach. In Section 5, we evaluate the application of our approach in BSH and CAF. In Section 6, we examine the related work of the area. Finally, we present our conclusions in Section 7.

2 BACKGROUND

In this section, we present the Domain Specific Language (DSL) that is used by BSH to formalize their products and the models at runtime of the induction hobs to which we apply our approach.

The newest Induction Hobs (IHs) [1] feature full cooking surfaces, where dynamic heating areas are automatically generated and activated or deactivated depending on the shape, size, and position of the cookware placed on the top. These dynamic areas are managed at runtime by calculating the resulting model after the changes in the context of the IH.

The Domain Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 metaclasses, 47 references among them, and more than 180 properties. For legibility reasons and due to intellectual property right concerns, in this section, we show a simplified subset of the IHDSL (see Fig. 1, IHDSL Metamodel and IHDSL Syntax). However, the evaluation was performed using the full IHDSL that is used in BSH. The Product Model in Figure 1 depicts an example of a product model that is specified with the IHDSL.

Inverters are in charge of transforming the electric supply to match the specific requirements of the IH. Then, the energy is transferred to the inductors through the channels. There can be several alternative channels, which enable different heating strategies depending on the cookware placed on top of the IH at runtime. The path followed by the energy through the channels is controlled by the power manager. Inductors are the elements where the energy is transformed into an electromagnetic field.

Figure 2 shows the behavior of an Induction Hob at runtime [2]. The IH is turned on in an initial configuration with a known model. In the face of changes in the context (CCs in Figure 2), reconfigurations (Rs in Figure 2) are triggered in order to change the configuration of the IH. Then, the Induction Hob is in a different configuration and therefore in a different model (Model Configurations in Figure 2). Some examples of relevant context changes include putting a pot on top, the pot reaches the set temperature, the pot is moved to other place on the IH, or liquid spills from the pot onto the surface. The reconfigurations activate or deactivate inductors and inverters and connect them through channels.

[1]https://www.youtube.com/watch?v=HjZ_nB-TY7w
[2]https://www.youtube.com/watch?v=Gp6urUZZbek

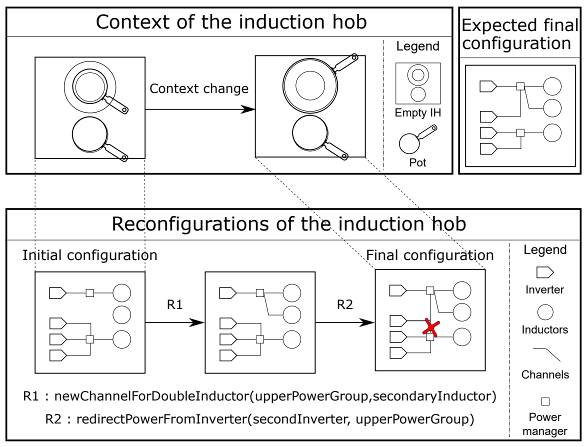

Context of the induction hob
Context change
Legend
Empty IH
Pot
Expected final configuration
Reconfigurations of the induction hob
Initial configuration
R1
R2
Final configuration
Legend
Inverter
Inductors
Channels
Power manager
R1 : newChannelForDoubleInductor(upperPowerGroup,secondaryInductor)
R2 : redirectPowerFromInverter(secondInverter, upperPowerGroup)

Figure 3: Example of a context change, a bugged reconfiguration and the reconfiguration rules performed

3 MOTIVATION

Figure 3 shows a reconfiguration that occurs in the Induction Hobs of BSH. In the initial configuration, the Induction Hob has two pots on top, heated through two inductors. The upper inductor is powered with one inverter and the bottom inductor is powered with three inverters. When a bigger pot is placed in the upper inductor, the Induction Hob reconfigures itself. Another inductor is activated ('R1'), and more energy is needed to heat the pot ('R2'). Therefore, the second inverter should give power to the upper inductors. However, the second inverter is not disconnected from the bottom power group (see the cross in Figure 3). This situation causes a bug, because when the user changes the power level of the upper inductors, the same power level will be applied to the bottom inductors, and vice versa. This bug was solved by modifying the reconfiguration rule 'R2' so that in addition to redirecting the power of the inverter, the inverter is also disconnected from the previous power group (see expected final configuration in Figure 3).

While the system is in use, any reconfiguration can be activated at any time. An Induction Hob can have a useful life of more than ten years, so the number of different combinations of reconfigurations that can be triggered is very high. In addition, some commercial Induction Hobs can have up to 48 inductors that dynamically connect and disconnect from the inverters. This Induction Hob has 2^{48} possibilities for activating or deactivating the inductors, beside differences in activation and deactivation orders. In addition, inductors and inverters are not the only dynamic components of the Induction Hob. Since the search space is too large, it is impossible to explore the space of possibilities exhaustively. Therefore, we use an evolutionary algorithm to locate bugs in the reconfiguration rules.

4 BUG LOCALIZATION IN RECONFIGURATION RULES

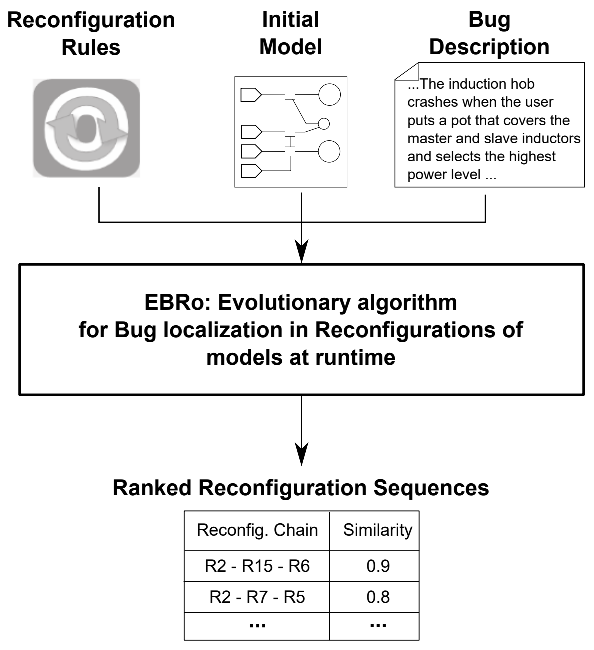
This section describes how the issue of bug localization in the reconfigurations of runtime models can be addressed by using our evolutionary algorithm, and the principles of our proposed method. Therefore, we first present an overview of our approach and, subsequently, provide the details of the approach and our adaptation of the evolutionary algorithm.

4.1 Approach Overview

The general structure of our approach (EBRo) is introduced in Figure 4. The goal of EBRo is to obtain a ranked list of sequences of reconfigurations rules from a given list of reconfiguration rules that may trigger the bug specified by the bug description. Our EBRo approach takes the following inputs:

- A set of reconfiguration rules that describe the changes in the model at runtime. The reconfiguration rules are triggered by context changes.
- An initial model which is the model that specifies the initial configuration. In our case, the configuration when the induction hob is turned on.
- A bug description of the target bug, using natural language. Typically, the description comes from textual documentation of a bug report. Therefore, the query will include some domain specific terms that are similar to those used when specifying the reconfiguration rules and the models. The knowledge of the engineers about the domain and the reconfiguration rules and the models will be useful for selecting the description from the bug report.

The output of EBRo (see Figure 4) is an ordered set of reconfiguration sequences that might trigger the target bug. The ranking is ordered following the similarity to the bug description. The search space for our approach is determined not only by the number of triggered reconfigurations, but also by the order in which they are applied. To explore the search space, EBRo uses an evolutionary


Reconfiguration Rules
Initial Model
Bug Description
EBRo: Evolutionary algorithm for Bug localization in Reconfigurations of models at runtime
Ranked Reconfiguration Sequences

...The induction hob crashes when the user puts a pot that covers the master and slave inductors and selects the highest power level ...

Reconfig. Chain	Similarity
R2 - R15 - R6	0.9
R2 - R7 - R5	0.8
...	...

Figure 4: Input and output of our bug localization in reconfigurations of models at runtime

algorithm that enables the exploration of a large number of possible reconfigurations. The evolutionary algorithm and its adaptation to the bug localization problem are described in the following sections.

4.2 Adapting the Evolutionary Algorithm for Bug Localization in Reconfigurations of Models at Runtime

Evolutionary algorithms are inspired by Darwin's evolutionary theory, where a population of individuals is modified through crossover and mutation operators [9]. Hence, to develop an evolutionary algorithm, the following elements have to be defined:

- Representation of the individuals.
- Evaluation of the individuals using a fitness function for each objective to determine a quantitative measure of their ability to solve the problem under consideration.
- Selection of the individuals to transmit from one generation to another.
- Creation of new individuals using genetic operators (crossover and mutation) to explore the search space.

The following paragraphs describe the design of these elements for our evolutionary algorithm for bug localization in reconfigurations of models at runtime.

4.2.1 Individual representation. To represent a candidate solution (individual), we used a vector representation. Each vector's dimension represents a reconfiguration rule. Thus, a solution is defined as a sequence of reconfigurations applied to a model. The size of the solution represents the number of reconfigurations (dimensions) in the vector. When created, the order of the reconfigurations corresponds to their positions in the vector.

R1: newChannelForDoubleInductor(upperPowerGroup, secondaryInductor)
R21: redirectPowerFromInverter(secondInverter,upperPowerGroup)
R3: activatePowerFromInverter(secondInverter)

Figure 5: Representation of an individual

An example of an individual is given in Figure 5. This individual contains three dimensions that correspond to three reconfigurations applied to the initial model. For instance, the predicate *newChannelForDoubleInductor(upperPowerGroup, secondaryInductor)* means that a new *channel* is created in the *upper power group*, connecting it with the *secondary inductor.*

4.2.2 Fitness function. After creating a solution, it should be assessed using a fitness function. The fitness function quantifies the quality of the proposed reconfiguration sequence. In our work, we use an information retrieval technique called Latent Semantic Indexing (LSI). Our algorithm assesses the relevance of each reconfiguration sequence in relation to the bug description provided by the user. The input of this step is a set of reconfiguration sequences, and the output is the set of reconfiguration sequences, where each reconfiguration sequence has been assigned with a fitness value regarding its similarity to the bug description.

To assess the relevance of each reconfiguration sequence in relation to the bug description provided by the user, we apply methods based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Indexing (LSI) [20, 32] to analyze the relationships between the description of the bug provided by the user and the reconfiguration sequences. There are many IR techniques, but most of the efforts show better results when applying LSI [20, 30, 32], specially when working with source code. Models are representations at a higher abstraction level than the source code, and the language used to build them is closer to the bug description language; therefore, we expect it to work better than when applied to source code.

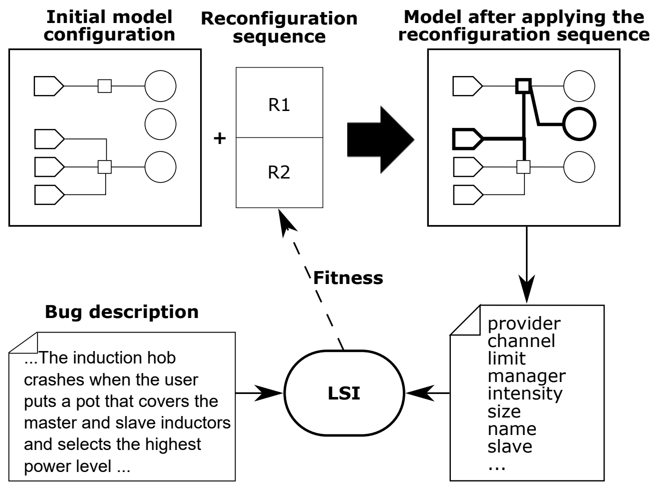

Initial model configuration
Reconfiguration sequence
Model after applying the reconfiguration sequence
R1
R2
+
Fitness
Bug description
...The induction hob crashes when the user puts a pot that covers the master and slave inductors and selects the highest power level ...
LSI
provider
channel
limit
manager
intensity
size
name
slave
...

Figure 6: Terms extraction from a reconfiguration sequence

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

Figure 6 shows how we extract the texts needed to use LSI. First, we apply the reconfiguration sequence to the initial model configuration. After applying it, we obtain a new model from which we extract the model elements that have been modified by the reconfigurations. In Figure 6, the modified model elements are the ones in bold. The texts for the LSI documents are the names and values of the properties and methods of each model element.

In our work, the LSI documents are model elements, i.e., a document of text is generated from the text of the model elements that have been modified by the reconfiguration. The query is constructed from the text that appears in the bug description. If the terms used for the model elements and the bug description differ too much, the LSI will not work. Therefore, the text from the documents (model elements) and the text from the query (bug description) are homogenized by applying well-known Natural Language Processing techniques (tokenizing, Parts-of-Speech Tagging, and Lemmatizing) to reduce this gap. If the languages used differ too much, other techniques such as manual annotation of the model elements could be applied at the expense of increasing the effort.

The union of all the keywords extracted from the documents (model elements) and from the query (bug description) are the terms (rows) used by our LSI fitness. Each column is one of the model elements that have been modified by the reconfiguration. The last column is the query obtained from the bug description of the user. Each row is one of the terms extracted from the corpuses of text composed by all of the model elements and the query itself. Each cell has the number of occurrences of each of the terms in the model elements.

Once the matrix is built, we normalize and decompose it into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [18]. One vector that represents the latent semantics of the document is obtained for each model fragment and the query. Finally, the similarities between the query and each model fragment are calculated as the cosine between the two vectors. The fitness value that is given to each model fragment is the one that we obtain when we calculate the similarity, obtaining values between -1 and 1.

4.2.3 *Selection.*

To select individuals, we use stochastic universal sampling (SUS) [5]. This technique of selection of an individual is directly proportional to its relative fitness in the population. SUS is a random selection algorithm which gives a higher probability of selection to the fittest solutions while still giving a chance to every solution.

In each iteration of the algorithm, SUS is used to select individuals from the population (P_n) for the next generation of the population (P_{n+1}). The selected individuals will be the ones that generate the next individuals using genetic operations.

4.2.4 *Genetic operators.*

To better explore the search space, the crossover and mutation operators are defined:

- Crossover: we use a single, random, cut-point crossover. It starts by selecting and splitting at random two parent solutions. When two parent individuals are selected, a random cut point is determined to split them into two sub-vectors. Then, the crossover creates two child solutions by putting, for the first child, the first part of the first parent with the second part of the second parent, and, for the second child, the first part of the second parent with the second part of the first parent.
  Each solution has a length limit in terms of number of reconfigurations. When applying the crossover operator, the new solution may have the minimum length between the two parents. Then, the crossover operator must enforce the length limit constraint by eliminating some reconfiguration rules.
  Figure 7 shows an example of applying the crossover operator. In this example, Parent 1 (P_1) and Parent 2 (P_2) are combined to generate two new solutions. The upper sub-vector of P_1 is combined with the bottom sub-vector of P_2 to form Child 1, and the bottom sub-vector of P_1 is combined with the upper sub-vector of P_2 to form Child 2.
- Mutation: This operator consists of randomly changing one or more reconfigurations in the vector of reconfigurations. Given an individual, the mutation operator first randomly selects some positions in the vector representation of the individual. Then, the selected dimensions are replaced by other reconfiguration rules.
  Figure 7 shows an example of applying the mutation operator. In Child 1, the mutation operator replaces dimension number two (*R5* by *R12*), while in Child 2, the mutation operator replaces dimensions number one and three (*R14* and *R3* by *R2* and *R6*).

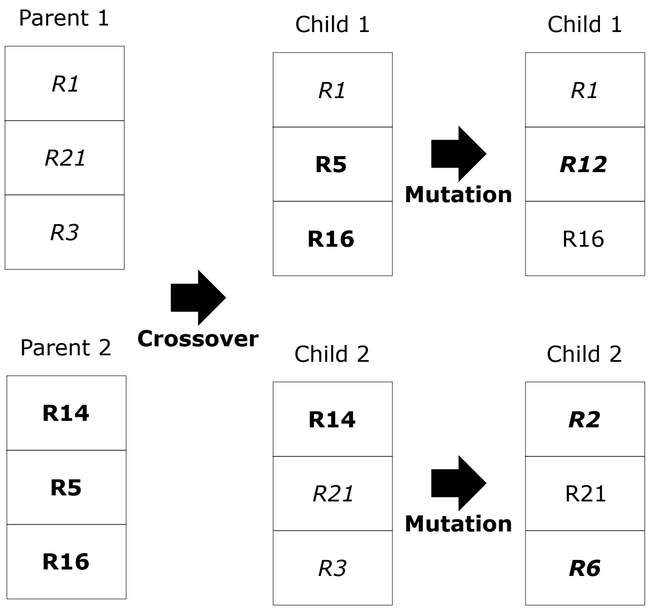


Figure 7: Crossover and mutation operators applied to reconfigurations

When creating the sequence of reconfigurations, we do not guarantee that they are feasible and that they can be applied. However, this could be solved by applying some repair operations that are out of the scope of this paper.

As a result, new reconfiguration sequences are created. In other words, the new reconfiguration sequences represent other possible solutions that can trigger the bug for the specific bug being located.

Overall, the aim of the approach is to find the most relevant reconfiguration sequence that triggers the bug described by the bug report. To do so, the algorithm of EBRo performs a search guided by a fitness function. This search is done among the different reconfiguration sequences (previously obtained by applying the mutation and crossover operations) that could conform to the bug description.

5 EVALUATION

This section presents the evaluation of our approach: the oracle preparation, the experimental setup, the description of the case studies where we applied the evaluation, the obtained results, the statistical analysis, the discussion of the results, and the threats to the validity of our work.

To evaluate the approach, we applied it to two long-living industrial case studies from two of our partners: BSH, the leading manufacturer of home appliances in Europe; and CAF, an international provider of railway solutions all over the world.

5.1 Oracle preparation

The oracle is the ground truth and is used to compare the results provided by the EBRo approach, the baseline, and a random search (RS) that works as sanity check. The baseline is the approach used by our industrial partner for bug localization [14]. A bug can be seen as an unwanted functionality, and a feature represents a functionality. For this reason, the feature localization approach presented in Font et. al [14] can be used for bug localization. Even though it was designed with a more general purpose in mind (feature localization), said approach is the best bug localization technique available to our industrial partner.

To prepare the oracle, our industrial partner provided us with the bug reports associated to bugs that have occurred in the product models. These bug reports contain natural language bug descriptions, the approved reconfigurations that trigger the target bugs and the model fragments that contain the bugs.

5.2 Experimental setup

This experiment evaluates whether or not the information found in the reconfiguration sequences improves the bug localization results. In addition, we compare the EBRo approach with the baseline [14] and with a random search (RS) sanity check. If RS outperforms an intelligent search method, we can conclude that there is no need to use a metaheuristic search.

The inputs of the evaluation process provided by our industrial partner are the models of the induction hobs, the entire set of reconfiguration rules, and the bug reports. The models, reconfiguration rules, and bug descriptions are used to run the EBRo and RS approaches, while the models and descriptions are used to run the baseline approach. We run each of the approaches and obtain a ranking of solutions that we can compare with an oracle in order to check accuracy. From the EBRo and RS approaches, we obtain reconfiguration sequences as solutions while from the baseline, we obtain model fragments as solutions.

Therefore, in order to compare the baseline and RS approaches against EBRo, we take the best solutions from the three approaches and compare them to the actual solution (from the oracle) that contains the trigger of the target bug in order to get a confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case, EBRo, the baseline, and RS) on a set of test data (the solutions) for which the true values are known (from the oracles). In our case, each solution that is output by the EBRo and the RS approaches is a sequence of reconfigurations composed of a subset of reconfigurations that are present in the sequence that triggers the bug. Since the granularity will be at the level of reconfigurations, the presence or absence of each reconfiguration rule will be considered as a classification. In the same way, each solution outputted by the baseline approach is a model fragment composed of a subset of the model elements that are present in the product model (where the bug is being located). Since the granularity will be at the level of model elements, the presence or absence of each model element will be considered as a classification. Therefore, our confusion matrices will distinguish between two values: TRUE (presence), and FALSE (absence).

The comparison process contrasts a result from one of the evaluated approaches with the ground truth from the oracle. We obtain a confusion matrix for each of the solutions predicted by each of the approaches. The confusion matrix arranges the results of the comparison into four categories:

- True positive (TP): an element present in the predicted solutions that is also present in the actual solution,
- True Negative (TN): an element not present in the predicted solution that is not present in the actual solution,
- False Positive (FP): an element present in the predicted solution that is not present in the actual solution, and
- False Negative (FN): an element not present in the predicted solution that is present in the actual solution.

The confusion matrix holds the results of the comparison between the predicted results and the actual results. The result of the sum of all the categories (TP+TN+FP+FN) is the number of dimensions (n) of the vector that contains the predicted solution. However, in order to evaluate the performance of the approach, it is necessary to extract some measurements from the confusion matrix. Therefore, some performance measurements are derived from the values in the confusion matrix. Specifically, we create a report that includes four performance measurements (recall, precision, F-measure, and MCC) for each of the test cases for EBRo, the baseline, and the RS approach.

Recall measures the number of elements of the solution that are correctly retrieved by the proposed solution and is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle) and is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

F-measure corresponds to the harmonic mean of precision and recall and is defined as follows:

$$F{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

Recall values can range between 0% (i.e., no single element from the actual solution is present in the predicted solution) to 100% (i.e., all the elements from the actual solution are present in the predicted solution).

Precision values can range between 0% (i.e., no single element from the predicted solution is present in the actual solution) to 100% (i.e., all the element from the predicted solution are present in the actual solution). A value of 100% precision and 100% recall implies that both the predicted solution and the actual solution are the same.

However, none of these measures correctly handle negative examples (TN). MCC is a correlation coefficient between the observed

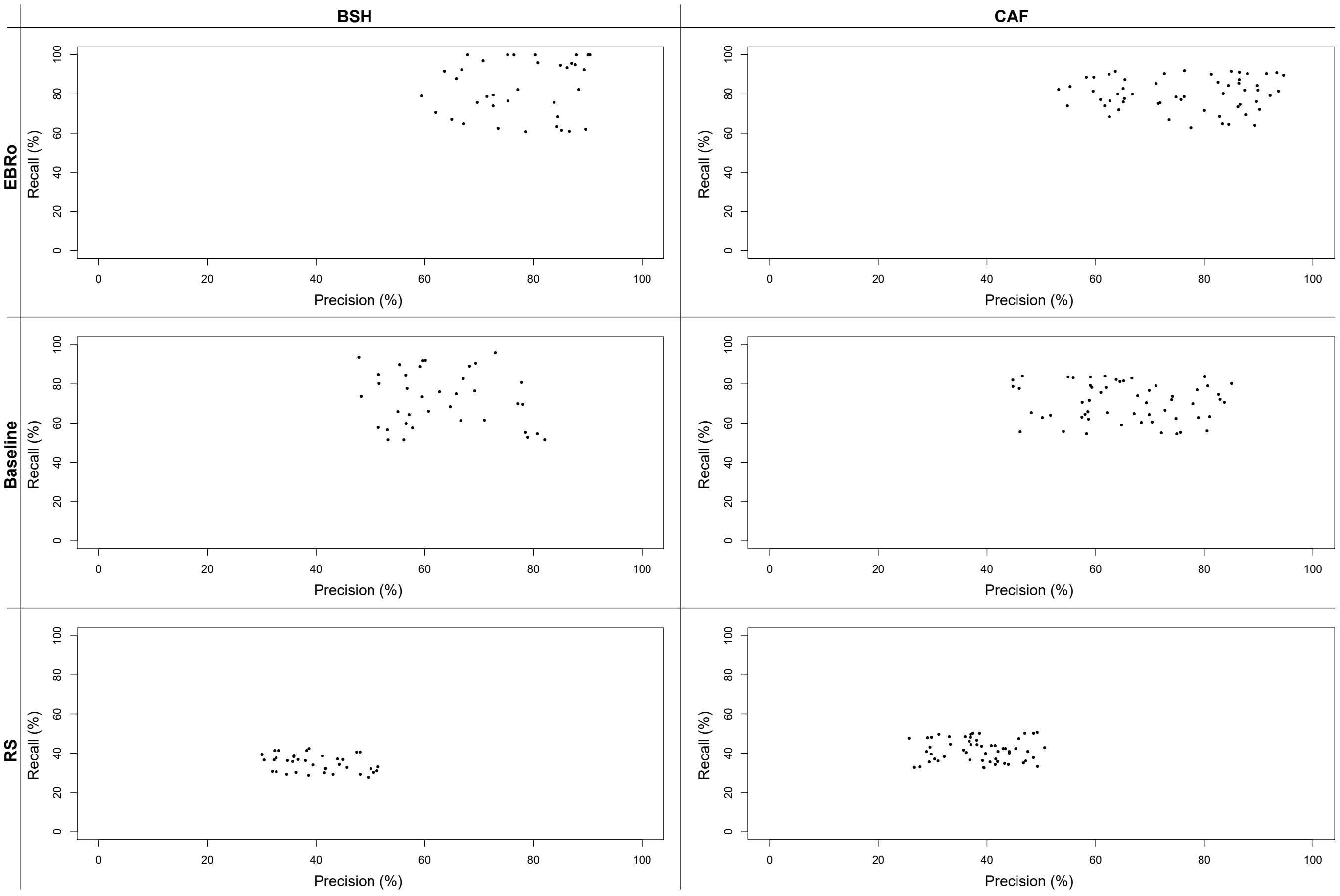


Figure 8: Mean recall and precision for EBRo, the baseline and the RS approaches in BSH and CAF

and predicted binary classifications that takes into account all of the observed values (TP, TN, FP, FN). MCC is a balanced measure which can be used even if the search space and the predicted solution are of very different sizes [8]. For this reason, MCC is one of the best measures for describing a confusion matrix [31]. It is defined as follows:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4}$$

5.3 Case studies

5.3.1 BSH. The first case study where we apply our evaluation process is BSH (already presented in Section 2). The oracle is composed of 46 induction hob models and its revisions over time where, on average, each product model is composed of more than 500 elements. Our industrial partner provided us with documentation of 37 bug reports, the approved reconfiguration sequences that triggers the bugs and the model fragments that contain the bugs. For each of the 37 bugs, we created a test case that includes the initial model, the set of reconfiguration rules and the model fragment where that bug was manifested and a bug description, all obtained from the documentation.

For this case study, we executed 30 independent runs for each of the 37 test cases for each approach (as suggested by [4]), i.e., 37 (bugs) * 3 (approaches) * 30 repetitions = 3330 independent runs.

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

The DSL of our industrial partner has the required expressiveness to describe the interaction between the main pieces of equipment installed in a train unit and their reconfigurations at runtime. Examples of reconfigurations at runtime can be coupling (when two trains are coupled together to increase the transport capacity or to rescue a train that is damaged) or converter assistance (allow the passing of current from one converter to equipment assigned to its peers in case of system overload or failure).

Again, we extract an oracle that is composed of 23 trains (and its revisions over time) where each product model is composed of more than 1200 elements on average. They provided us with documentation of 56 bug reports, the approved reconfiguration sequences that triggers the bugs and the model fragments that contain the bugs. For each of the 56 bugs, we created a test case that includes the initial model, the set of reconfiguration rules and the model fragment where that bug was manifested and a bug description, all obtained from the documentation.

For this case study, we executed 30 independent runs for each of the 56 test cases for each approach (as suggested by [4]), i.e., 56 (bugs) * 3 (approaches) * 30 repetitions = 5040 independent runs.

5.4 Results

In this section, we present the results obtained for each case study in EBRo, the baseline and the RS. Figure 8 shows the charts with the recall and precision results for the industrial domain of BSH (left column) and CAF (right column). A dot in a graph represents the average result of precision and recall for each bug (37 bugs in BSH and 56 bugs in CAF) for the 30 repetitions. The first row of charts shows the results of EBRo, the second row of charts shows the results of the baseline approach, and the third row of charts shows the results of the RS approach.

Table 1: Mean values and standard deviations for precision, recall, F-Measure and MCC for each approach and each case study

		EBRo	Baseline	RS
BSH	Recall ± (σ)	83.40 ± 14.28	72.51 ± 14.07	35.19 ± 4.30
	Precision ± (σ)	78.20 ± 9.41	63.34 ± 9.95	40.22 ± 6.52
	F-measure ± (σ)	79.96 ± 9.33	66.43 ± 7.47	37.05 ± 3.07
	MCC ± (σ)	0.78 ± 0.10	0.63 ± 0.08	0.31 ± 0.04
CAF	Recall ± (σ)	80.12 ± 8.22	70.94 ± 9.52	41.87 ± 5.66
	Precision ± (σ)	76.35 ± 12.14	65.55 ± 11.26	38.81 ± 6.64
	F-measure ± (σ)	77.49 ± 7.81	67.26 ± 7.37	39.79 ± 4.44
	MCC ± (σ)	0.75 ± 0.08	0.64 ± 0.08	0.33 ± 0.05

Table 1 shows the mean values of recall, precision, F-measure, and MCC for the EBRo, baseline, and RS approaches in each case study. The EBRo approach obtains the best results in recall, precision, and MCC, providing an average value of 83.40% in BSH and 80.12% in CAF in recall, 78.20% in BSH and 76.35% in CAF in precision, and 0.78 in BSH and 0.75 in CAF in MCC. The second best results are obtained by the baseline, providing an average value of 72.51% in BSH and 70.94% in CAF in recall, 63.34% in BSH and 65.55% in CAF in precision, and 0.63 in BSH and 0.64 in CAF in MCC. The RS approach provided an average value of 35.19% in BSH and 41.87% in CAF in recall, 40.22% in BSH and 38.81% in CAF in precision, and 0.31 in BSH and 0.33 in CAF in MCC. In terms of recall, precision, and MCC, EBRo outperforms the rest of the approaches.

5.5 Statistical Analysis

Statistically significant differences can be obtained even if they are so small as to be of no practical value [4]. The goals of our statistical analysis are: (1) to provide formal and quantitative evidence (statistical significance) that EBRo does in fact have an impact on the comparison metrics (i.e., that the differences in the results were not obtained by mere chance); and (2) to show that those differences are significant in practice (effect size).

5.5.1 Statistical significance. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. The Quade test shows that it is more powerful than the others when working with real data [15]. This tests provide a probability value, $p - Value$. The $p - Value$ obtains values between 0 and 1. It is accepted by the research community that a $p - Value$ under 0.05 is statistically significant [4]. The $p - Values$ obtained in the test are well below 0.05. Consequently, we can state that there are significant differences among the algorithms for the four performance indicators.

In addition, we perform a post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each algorithm, determining whether statistically significant differences exist among the results of a specific pair of algorithms. Specifically, we apply the Holm's Post Hoc analysis, as suggested by Garcia et al. [15].

Table 2: The $p - Values$ of Holm's post hoc analysis for each pair of algorithms and each case study

		EBRo vs Baseline	EBRo vs RS	Baseline vs RS
BSH	Recall	0.022	$1.1x10^{-14}$	$1.7x10^{-10}$
	Precision	$2.9x10^{-7}$	$\ll 2.2x10^{-16}$	$5.2x10^{-10}$
	MCC	$2.9x10^{-8}$	$\ll 2.2x10^{-16}$	$3.0x10^{-10}$
CAF	Recall	$3.8x10^{-5}$	$\ll 2.2x10^{-16}$	$9.2x10^{-15}$
	Precision	$3.0x10^{-5}$	$\ll 2.2x10^{-16}$	$1.2x10^{-14}$
	MCC	$1.4x10^{-10}$	$\ll 2.2x10^{-16}$	$1.4x10^{-14}$

Table 2 shows the $p - Values$ of Holm's post hoc analysis. All of the $p - Values$ shown in this table are smaller than their corresponding significance threshold value (0.05), indicating that the differences in performance between the algorithms are significant.

5.5.2 Effect size. For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [37]. $\hat{A}_{12}$ measures the probability that running one approach yields higher values than running another approach. If the two approaches are equivalent, then $\hat{A}_{12}$ will be 0.5.

Table 3 shows the values of the size effect statistics. The largest differences were obtained between EBRo and the RS approach, where EBRo achieves better results all of the times. When comparing EBRo and the baseline, the differences are not so large, with EBRo achieving better results in around 80% of the times. EBRo obtained the best performance results among the three evaluated approaches (see Table 1).

The performed statistical analysis indicated that EBRo outperforms the rest of the approaches in terms of recall, precision, and MCC. Overall, these results confirm that the use of EBRo against the baseline and the RS approaches has an actual impact.

Table 3: The $\hat{A}_{12}$ statistic for each pair of algorithms and each case study

		EBRo vs Baseline	EBRo vs RS	Baseline vs RS
BSH	Recall	0.7166	1	1
	Precision	0.8524	1	0.9912
	MCC	0.8703	1	1
CAF	Recall	0.7548	1	1
	Precision	0.7411	1	0.9825
	MCC	0.8324	1	0.9992

5.6 Discussion

Our results confirm that the EBRo and the baseline approaches are better than random search based on the four metrics (recall, precision, F-measure, and MCC) on both the BSH and CAF case studies. Through this study, we concluded that there is empirical evidence to support the significance of the results of our algorithm. Thus, an intelligent algorithm is required to find good solutions to perform bug localization in reconfigurations of models at runtime.

In addition, the solutions obtained with the EBRo approach are better than the ones obtained with the baseline approach. The search space with the EBRo approach is driven by the reconfigurations, as it happens at runtime. While the baseline explores a much larger search space, any model conforms to the metamodel, and does not take into account the runtime reconfigurations.

However, the EBRo and baseline approaches are complementary. In the real world, before locating a bug, in most cases we do not know if the reconfigurations are the source of the bug for sure. Faced with a new bug, our recommendation is to start searching with EBRo, since EBRo not only obtains the configuration of the model where the bug occurs, but also provides the sequence of reconfigurations that leads to that particular configuration. In case the results obtained by EBRo are not useful, the baseline can be launched, since it explores a larger solution space than EBRo.

We realized that none of the approaches obtain a perfect solution. One of the issues that we detected that cause this outcome is the vocabulary mismatch. That means that for a specific concept, the terms used in the bug description are different from the terms used in the models. Nevertheless, this issue could be solved by augmenting the Natural Language Processing (NLP) with a dictionary of synonyms. In the same way, we have also detected cases in which in-house terms are used. Therefore, the regular dictionary of synonyms would not work in this case. This suggests that the dictionary of synonyms should be refined by domain engineers to include in-house terms. Another issue occurs when a bug description is incomplete. Omitting words in the bug descriptions negatively influences the fitness value of textual similarity since the fitness value of textual similarity is based on the co-occurrence of terms. This suggests that we must make the engineers aware of this issue. They should know that in cases in which the results obtained do not have enough quality, they can reformulate the descriptions of the bugs making the implicit knowledge explicit.

In addition, some solutions are invalid sequences of reconfigurations that, theoretically, are not going to take place since the necessary context changes can never occur. In our future work, we will study the introduction of crossover and mutation operators that, by construction, do not generate invalid sequences. We will also study a repair operation for invalid sequences. Although we are interested in the influence of narrowing the search space, we believe that we must maintain the possibility of generating sequences of theoretically invalid reconfigurations. For instance, induction hobs are sold all over the world and sometimes they are used in unforeseen ways, causing extremely unlikely sequences of reconfigurations to end up happening. These otherwise impossible reconfigurations are potential sources of bugs.

Results suggest that there is a relationship between the use of models at runtime and the quality of the solutions. The results are better the more you have specified the reconfigurations using the models at runtime. That is, the fewer flow control operators (e.g., ifs) and auxiliary variables are used to reconfigure, the better. This finding suggests that the models at runtime pay off when doing bug localization. However, the finding must be taken cautiously, since our evaluation was not designed to verify the influence of models at runtime on the quality of the solutions, but to compare the performance of different approaches in the location of bugs in the reconfigurations of models at runtime.

Finally, we have applied our EBRo approach to two real industrial cases, BSH and CAF, which are from very different domains. In both scenarios we have obtained quality results. Hence, the findings suggest that EBRo does not rely on the domain, since results depend on the reconfigurations themselves.

5.7 Threats to Validity

In this section, we present some of the threats to validity. We follow the guidelines suggested by De Oliveira et. al [11] to identify those that are applicable to this work.

Conclusion validity threats: We have identified three threats of this type. To address the *not accounting for random variation* threat, we considered 30 independent runs for each bug with each algorithm. In this paper we employed standard statistical analysis following accepted guidelines [4] to avoid the *lack of formal hypothesis and statistical tests* threat. To address the *lack of good descriptive analysis* threat, we have used precision, recall, F-measure and MCC metrics to analyze the confusion matrix obtained from the experiments; however, other metrics could be applied. In addition, some works argument that the use of the Vargha and Delaney A12 metric can be miss-representative [28] and that the data should be pre-transformed before applying them. We did not find any use case for data pre-transformation that applies to our case study.

Internal validity threats: We have identified two threats of this type. In this paper, we used standard values for the algorithms to avoid the *poor parameter settings* threat. These values have been tested in similar algorithms for feature localization [22]. In addition, the choice of the k value in the application of SVD can produce suboptimal accuracy when using LSI for software artifacts [29]. However, we plan to evaluate all of the parameters of our algorithm in a future work. The evaluation of this paper was applied to two industrial case studies from two of our partners, BSH and CAF, hence the *lack of real problem instances* threat is addressed.

Construct validity threats: We have identified one threat of this type. To address the *lack of assessing the validity of cost measures* threat, we performed a fair comparison among EBRo, the baseline and the random approaches by generating the same number of reconfiguration sequences and using the same number of fitness evaluations.

External validity threats: We have identified two threats of this type. The *lack of a clear object selection strategy* and *lack of evaluations for instances of growing size and complexity* threats are addressed by using two industrial case studies from two of our partners, BSH and CAF. Our instances are collected from real-world problems. In addition we have two different domains (induction hobs and trains) with different size and complexity.

6 RELATED WORK

In recent years, the research efforts in Search-Based Software Engineering (SBSE) in models at runtime are increasing.

McKinley et al. [26] applied SBSE to address uncertainty in adaptive systems. They integrated evolutionary computation into the development and runtime support of high-assurance, self-adaptive software. Their approach starts with requirements and moves through reconfigurable designs at runtime. Their work has been validated using experiments conducted in the context of robotics.

Andrade and de A Macêdo [1] have proposed a search-based approach for domain-independent representation of design spaces. Their approach (DuSE-MT) capture the most prominent design dimensions, their associated variation points, and the architecture changes requires to realize each solution. Their solutions are evaluated in terms of four quality metrics related to self-adaptation.

Williams et al. [39] presented a model-driven approach for adapting to dynamic runtime environments using metaheuristic optimization techniques. The metaheuristics exploit metamodels that capture the important components in the adaptation process. They contextualize the approach using an example and analyze different ways of applying the metaheuristic algorithms for discovering an optimal model of the case study's environment.

All of the above works present approaches that combine SBSE and models at runtime. However, these research efforts are focused on finding the optimal reconfigurations. In contrast, our work is focused on localizing bugs appearing as the result of dynamic reconfigurations of the system due to context changes.

In addition, there are other SBSE approaches for feature localization in models that, although not designed to locate bugs, could potentially be applied to that extent.

Lopez-Herrejon et al. [22] evaluate three standard search-based techniques with three objective functions in order to calculate the relationships of a feature model. Their results are slightly better for hill climbing than for the evolutionary algorithm, but they are not statistically significant when the first two objective functions are applied.

Harman et al. [17] performed a survey on the topic of search-based software engineering applied to Software Product Lines (SPLs). They present an overview of recent articles that are classified according to themes such as configuration, testing, or architectural improvement. Lopez-Herrejon et al. [21] performed a preliminary systematic mapping study at the connection of search-based software engineering and SPL. They categorized the articles using a known framework for SPL development.

Font et al. [13, 14] propose two approaches that use evolutionary algorithms to locate features in a model. They introduce a genetic operator for mutation that can work with a single model fragment and a crossover operator that combines two different product models. The results show that the use of a genetic algorithm allows the approach to provide accurate location of features in spite of inaccurate information on the part of the user.

In contrast to our work, all these feature location works only take into account the design time models.

In our previous work [3], we analyzed the influence of several timespan weightings on bug localization in models. We evaluated four timespan weightings: the most recent model modifications, the oldest model modifications, the mean of the modification timespan of the modified model elements, and the sum of the modification timespan of the modified model elements. The results showed that the use of the most recent timespan model modifications provides the best results in bug localization. In contrast to our previous works, our approach now takes into account the reconfigurations of models at run-time that can trigger the bug. Our future work includes the introduction of timespan weightings to improve the bug localization in the reconfigurations of models at runtime.

7 CONCLUSIONS

Bug localization is a significant maintenance activity and in a system with models at runtime, the reconfigurations that the model undergoes at runtime due to context changes can also be a source of bugs. In this paper, we have proposed an approach for bug localization in the reconfigurations of models at runtime (EBRo). Our approach is based on search-based software engineering, iterating on the reconfigurations search space using an evolutionary algorithm.

We evaluated our EBRo approach in two industrial case studies, BSH (firmware of induction hobs) and CAF (control software of trains), and compared it with the approach that they are using for bug localization and with a random search as sanity check. We determined which approach produces the best results in terms of precision, recall, F-measure, and MCC. To do this, we applied the approaches in BSH and CAF.

Results show that the study of the reconfiguration of models at runtime pays off for bug localization. Specifically, our EBRo approach outperforms the baseline and the random search approaches in terms of precision, recall, and MCC, reaching average precision and recall values of about 80% in BSH and 77% in CAF. Furthermore, results also suggest that our approach can be applied in real world environments. Finally, the statistical analysis of the results provides evidence of their significance.

ACKNOWLEDGMENTS

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R). We also thank ITEA3 15010 REVaMP2 Project.

REFERENCES

[1] S. S. Andrade and R. J. de A Macêdo. 2013. Toward Systematic Conveying of Architecture Design Knowledge for Self-Adaptive Systems. In *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*. 23–24. DOI:http://dx.doi.org/10.1109/SASOW.2013.13

[2] Paolo Arcaini, Angelo Gargantini, and Elvinia Riccobene. 2012. CoMA: Conformance Monitoring of Java Programs by Abstract State Machines. In *Proceedings of the Second International Conference on Runtime Verification (RV'11)*. Springer-Verlag, 223–238. DOI:http://dx.doi.org/10.1007/978-3-642-29860-8_17

[3] Lorena Arcega, Jaime Font, Øystein Haugen, and Carlos Cetina. 2017. On the Influence of Modification Timespan Weightings in the Location of Bugs in Models. In *Proceedings of the 26th International Conference on Information Systems Development, ISD 2017, Larnaca, Cyprus, September 6-8, 2017.*

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

[5] James E. Baker. 1987. Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*. L. Erlbaum Associates Inc., 14–21. http://dl.acm.org/citation.cfm?id=42512.42515

[6] Nelly Bencomo, Robert France, Betty H. C. Cheng, and Uwe Aßmann (Eds.). 2014. *Models@run.time. Foundations, Applications, and Roadmaps.* Springer International Publishing.

[7] Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. Models@ Run.Time. *Computer* 42, 10 (Oct. 2009), 22–27. DOI:http://dx.doi.org/10.1109/MC.2009.326

[8] Sabri Boughorbel, Fethi Jarray, and Mohammed El-Anbari. 2017. Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. *PLOS ONE* 12, 6 (06 2017), 1–17. DOI:http://dx.doi.org/10.1371/journal.pone.0177678

[9] Ilhem Boussaïd, Patrick Siarry, and Mohamed Ahmed-Nacer. 2017. A survey on search-based model-driven engineering. *Automated Software Engineering* 24, 2 (01 Jun 2017), 233–294. DOI:http://dx.doi.org/10.1007/s10515-017-0215-4

[10] Paulo Casanova, Bradley Schmerl, David Garlan, and Rui Abreu. 2011. Architecture-based Run-time Fault Diagnosis. In *Proceedings of the 5th European Conference on Software Architecture (ECSA'11)*. Springer-Verlag, 261–277. http://dl.acm.org/citation.cfm?id=2041790.2041827

[11] Márcio de Oliveira Barros and Arilo Cláudio Dias-Neto. 2011. 0006/2011-Threats to Validity in Search-based Software Engineering Empirical Studies. *RelaTe-DIA* 5, 1 (2011).

[12] Ilenia Epifani, Carlo Ghezzi, Raffaela Mirandola, and Giordano Tamburrelli. 2009. Model Evolution by Run-time Parameter Adaptation. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*. IEEE Computer Society, 111–121. DOI:http://dx.doi.org/10.1109/ICSE.2009.5070513

[13] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. Feature Location in model-based Software Product Lines through a Genetic Algorithm. In *15th International Conference on Software Reuse (ICSR 2016)*. Limassol, Cyprus.

[14] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. 2016. Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)*. ACM, 272–282. DOI:http://dx.doi.org/10.1145/2976767.2976789

[15] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. 2010. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences* 180, 10 (2010), 2044 – 2064. DOI:http://dx.doi.org/10.1016/j.ins.2009.12.010 Special Issue on Intelligent Distributed Information Systems.

[16] Carlo Ghezzi, Andrea Mocci, and Mario Sangiorgio. 2012. Runtime Monitoring of Functional Component Changes with Behavior Models. In *Proceedings of the 2011th International Conference on Models in Software Engineering (MODELS'11)*. Springer-Verlag, 152–166. DOI:http://dx.doi.org/10.1007/978-3-642-29645-1_17

[17] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. 2014. Search Based Software Engineering for Software Product Line Engineering: A Survey and Directions for Future Work. In *Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC '14)*. ACM, 5–18. DOI:http://dx.doi.org/10.1145/2648511.2648513

[18] Thomas K Landauer, Peter W. Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse Processes* 25, 2-3 (1998), 259–284. DOI:http://dx.doi.org/10.1080/01638539809545028

[19] Meir M Lehman, JF Ramil, and Goel Kahen. 2001. *A paradigm for the behavioural modelling of software processes using system dynamics.* Technical Report. Imperial College of Science, Technology and Medicine, Department of Computing.

[20] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. 2007. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering (ASE '07)*. ACM, 234–243. DOI:http://dx.doi.org/10.1145/1321631.1321667

[21] Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Lukas Linsbauer, Alexander Egyed, and Enrique Alba. 2014. A Hitchhiker's Guide to Search-Based Software Engineering for Software Product Lines. *CoRR* abs/1406.2823 (2014).

[22] Roberto E. Lopez-Herrejon, Lukas Linsbauer, José A. Galindo, José A. Parejo, David Benavides, Sergio Segura, and Alexander Egyed. 2015. An assessment of search-based techniques for reverse engineering feature models. *Journal of Systems and Software* 103 (2015), 353 – 369. DOI:http://dx.doi.org/10.1016/j.jss.2014.10.037

[23] Usman Mansoor, Marouane Kessentini, Philip Langer, Manuel Wimmer, Slim Bechikh, and Kalyanmoy Deb. 2015. MOMM: Multi-objective model merging. *Journal of Systems and Software* 103 (2015), 423 – 439. DOI:http://dx.doi.org/https://doi.org/10.1016/j.jss.2014.11.043

[24] Shahar Maoz and David Harel. 2011. On tracing reactive systems. *Software & Systems Modeling* 10, 4 (01 Oct 2011), 447–468. DOI:http://dx.doi.org/10.1007/s10270-010-0151-2

[25] A. Marcus, A. Sergeyev, V. Rajlich, and J.I. Maletic. 2004. An information retrieval approach to concept location in source code. In *Proceedings of the 11th Working Conference on Reverse Engineering*. 214–223. DOI:http://dx.doi.org/10.1109/WCRE.2004.10

[26] Philip K. McKinley, Betty H. C. Cheng, Andres J. Ramirez, and Adam C. Jensen. 2012. Applying evolutionary computation to mitigate uncertainty in dynamically-adaptive, high-assurance middleware. *Journal of Internet Services and Applications* 3, 1 (01 May 2012), 51–58. DOI:http://dx.doi.org/10.1007/s13174-011-0049-4

[27] O. Moser, F. Rosenberg, and S. Dustdar. 2012. Domain-Specific Service Selection for Composite Services. *IEEE Transactions on Software Engineering* 38, 4 (July 2012), 828–843. DOI:http://dx.doi.org/10.1109/TSE.2011.43

[28] Geoffrey Neumann, Mark Harman, and Simon Poulding. 2015. *Transformed Vargha-Delaney Effect Size.* Springer International Publishing, 318–324. http://dx.doi.org/10.1007/978-3-319-22183-0_29

[29] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. 2016. Parameterizing and Assembling IR-Based Solutions for SE Tasks Using Genetic Algorithms. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 314–325. DOI:http://dx.doi.org/10.1109/SANER.2016.97

[30] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Transactions on Software Engineering* 33, 6 (June 2007), 420–432. DOI:http://dx.doi.org/10.1109/TSE.2007.1016

[31] D. M. W. Powers. 2011. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies* 2, 1 (2011), 37–63.

[32] M. Revelle, B. Dit, and D. Poshyvanyk. 2010. Using Data Fusion and Web Mining to Support Feature Location in Software. In *IEEE 18th International Conference on Program Comprehension (ICPC)*. 14–23. DOI:http://dx.doi.org/10.1109/ICPC.2010.10

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

[34] Daniel Schneider and Mario Trapp. 2013. Conditional Safety Certification of Open Adaptive Systems. *ACM Trans. Auton. Adapt. Syst.* 8, 2, Article 8 (July 2013), 20 pages. DOI:http://dx.doi.org/10.1145/2491465.2491467

[35] Hui Song, Michael Gallagher, and Siobhán Clarke. 2012. Rapid GUI Development on Legacy Systems: A Runtime Model-based Solution. In *Proceedings of the 7th Workshop on Models@Run.Time (MRT '12)*. ACM, 25–30. DOI:http://dx.doi.org/10.1145/2422518.2422523

[36] Michael Szvetits and Uwe Zdun. 2016. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling* 15, 1 (01 Feb 2016), 31–69. DOI:http://dx.doi.org/10.1007/s10270-013-0394-9

[37] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. DOI:http://dx.doi.org/10.3102/10769986025002101

[38] Thomas Vogel and Holger Giese. 2010. Adaptation and Abstract Runtime Models. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '10)*. ACM, 39–48. DOI:http://dx.doi.org/10.1145/1808984.1808989

[39] James R. Williams, Simon M. Poulding, Richard F. Paige, and Fiona Polack. 2013. *Exploring the Use of Metaheuristic Search to Infer Models of Dynamic System Behaviour.* 76–88.

[40] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (Aug 2016), 707–740.