On the Influence of Modification Timespan Weightings in the Location of Bugs in Models

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

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

Øystein Haugen *oystein.haugen@hiof.no*
Østfold University College, Department of Information Technology
Halden, Norway

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

Abstract

Bug location is a common task in Software Engineering, specially when maintaining and evolving software products. When locating bugs in code, results depend greatly on the way code modification timespans are weighted. However, the influence of timespan weightings on bug location in models has not received enough attention yet. Throughout this paper, we analyze the influence of several timespan weightings on bug location in models. These timespan weightings guide an evolutionary algorithm, which returns a ranking of model fragments relevant to the solution of a bug. We evaluated our timespan weightings in BSH, a real-world industrial case study, by measuring the results in terms of recall, precision, and F-measure. Results show that the use of the most recent timespan model modifications provide the best results in our study. We also performed a statistical analysis to provide evidence of the significance of the results.

Keywords: Bug Location, Model Driven Engineering, Reverse Engineering.

1. Introduction

During software evolution, the existing software of a project undergoes modifications to satisfy changes. A change may result in either the addition of a new software function, the removal of a bug or defect, or the improvement of an existing software functionality. These maintenance and evolution activities take up to 80% of the lifetime of a system [15]. Software maintainers spend from 50% up to almost 90% of their time trying to understand a program in order to make changes correctly [1]. One of the key issues to achieve this goal is finding relevant locations to address the changes.

Bug Location is one of the most important and common activities performed during software maintenance and evolution [4]. Currently, research efforts in Bug Location are concerned with identifying software artifacts associated with bug descriptions. However, most research on Bug Location targets code [23] as the software artifact that realizes the feature, neglecting other software artifacts such as models.

In order to locate bugs in code, the most recent code modifications are regarded as the most relevant. Bug location results depend greatly on the way in which the modification timespans are weighted. The consideration of timespans is based on the Defect Localization Principle. This principle is based on the observation that the most recent modifications to a project are most likely the cause of future bugs [10][30]. Considering recent project modifications, it is possible to find relevant code for bug location [26].

We perform Bug Location in Models (BLiM). To do so, we locate the most relevant model fragments for a particular bug description. Model fragments are formed by model elements, and each model element has an associated modification time. When we apply the Defect Principle to model fragments, we have to decide how to assign a modification time to the model fragment from the modification time information on its model elements. The contribution of this work is the design, application for BLiM, and evaluation of four fitness functions regarding modification timespan weightings. The weightings are the following: (1) the most recent model modifications (BLiM-recent), (2) the oldest model modifications (BLiM-oldest), (3) the mean of the modification timespan of the modified model elements (BLiM-mean), and (4) the sum of the modification timespan of the modified model elements (BLiM-sum).

In our evaluation, we have applied our approach to the product models from an industrial partner, BSH. We compare the results of running our BLiM approach with the different fitness functions. We measure the results using the standard information retrieval measurements: recall, precision, and the combination of both (F-measure) [25][18]. The outcome shows that the use of the most recent modification timespan of a model element as the modification timespan of a model fragment (BLiM-recent) provides the best results, and proves that the approach can be applied in real world environments. The statistical analysis of the results provides evidence of their significance.

The remainder of the paper is structured as follows: in Section 2, we present the Domain Specific Language used by the industrial partner. In Section 3, we describe our BLiM approach. In Section 4, we evaluate our approach with the data provided by the industrial partner. In Section 5, we examine the related work of the area. Finally, we present our conclusions in Section 6.

2. Background

The running example and the evaluation of this paper are performed through the products of the industrial partner, BSH. In this section, we present the Domain Specific Language (DSL) used by BSH to formalize their products, called IHDSL. In addition, we present the language used by our approach to formalize the model fragments, the Common Variability Language (CVL).

The newest Induction Hobs (IHs) 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. In addition, there has been an increase in the type of feedback provided to the user while cooking. All of these changes are being possible at the cost of increasing the complexity of the software behind IHs.

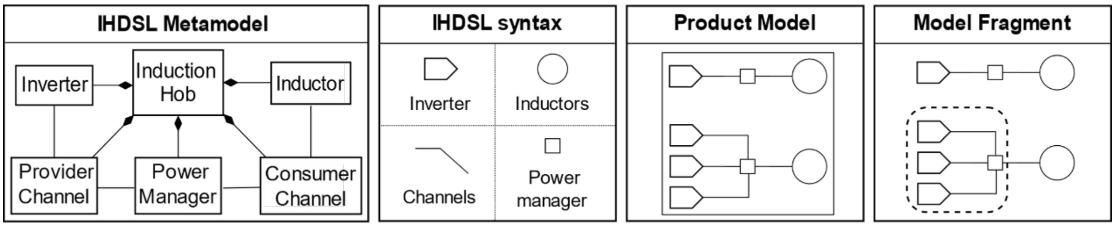


Fig. 1. IHDSL product model and model fragment formalization.

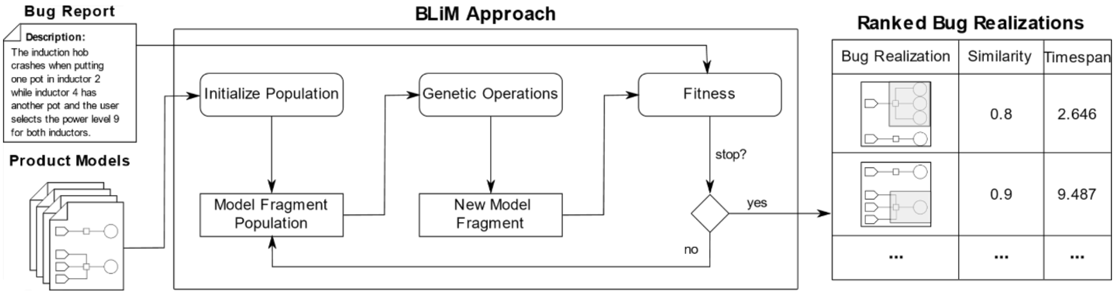
The Domain Specific Language used by BSH to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 47 references among them, and more than 180 properties. However, in order to gain legibility and due to intellectual property right concerns, in this paper we use a simplified subset of the IHDSL (see left part of Fig. 1, IHDSL Metamodel and IHDSL Syntax).

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

To formalize the model fragments used by the approach we use Common Variability Language (CVL) [11][27], given its capabilities to formalize a set of model elements as a model fragment. Right part of Fig. 1 shows an example of a model fragment of the product model. The model fragment includes the three inverters in charge of powering the lower inductor along with the three channels and the power manager used to aggregate and manage the power provided by those inverters.

3. Bug Location in Models (BLiM)

This section presents the BLiM approach for bug location. The left part of Fig. 2 shows an example of input for our approach. The approach receives as input a bug description of the bug that the software engineer wants to locate. Typically, these descriptions come 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 product models. In addition, the software engineer selects a set of product models from the entire family of products that include the bug to be located.



Description:
The induction hob crashes when putting one pot in inductor 2 while inductor 4 has another pot and the user selects the power level 9 for both inductors.

Bug Realization	Similarity	Timespan
	0.8	2.646
	0.9	9.487
...	...	...

Fig. 2. Overview of the Bug Location Approach in Models: BLiM.

The approach relies on an evolutionary algorithm. The center of Fig. 2 shows a simplified representation of the main steps. The 'Initialize Population' step calculates an initial population of model fragments from the input set of product models. This initial population of model fragments is randomly extracted from the product models. The 'Genetic Operations' produce the new generation of model fragments. First, a selection operation chooses the model fragments that will be used as parents of the new model fragments. The fitness values are used to ensure that the best model fragments are chosen as parents. Then, a crossover operation mixes the model elements of the two parents into a new model fragment. Finally, a mutation operation introduces variations in the new model fragment, in hopes that it achieves better fitness values than its parents. The 'Fitness' step assigns values that assess how good each model fragment is in the following terms: bug description and modification timespan.

As output, the approach provides a list of model fragments that might realize the bug. The output of BLiM (see the right part of Fig. 2) is a ranking of model fragments that realize the target bug. The ranking can be ordered following different criteria, such as the similarity of the model fragments to the bug description, or the model fragment modification timespans.

3.1. Genetic Operations of the BLiM Approach

The generation of model fragments is performed by applying genetic operators adapted to work on model fragments. In other words, new fragments are generated from existing ones through the use of two genetic operators: the crossover operator, and the mutation operator. We use the crossover and mutation operations presented in [6].

The crossover operation takes a model fragment from a first parent model and the whole product model from a second parent model, generating a new individual that contains elements from both parents and thus preserving the basic mechanics of the crossover operation.

The mutation possibilities of a given model fragment are driven by its associated product model. Each model fragment is associated to a product model, and the model fragment mutates in the context of its associated product model. In other words, the model fragment will gain or drop some elements, but the resulting model fragment will still be part of the referenced product model. For more details about these genetic operations see [6].

3.2. Fitness of the BLiM Approach

In evolutionary algorithms, the fitness step is used to imitate the different degrees of adaptation to the environment that different individuals have. Following this idea, our fitness step is used to determine the suitability of each model fragment to the problem. The input of this step is a population of model fragments, and the produced output is a set with each model fragment from the input population, accompanied by two fitness values: similarity to the feature description, and most recent model fragment modifications.

3.2.1. Model Fragment Similarity to the Bug Description

To assess the relevance of each model fragment in relation to the bug description provided by the user, we apply methods based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Analysis (LSA) [14] to analyze the relationships between the description of the bug provided by the user and the model fragments. There are many IR techniques, but most research efforts show better results when applying LSA [22] [16] [21].

LSA constructs vector representations of a query and a corpus of text documents by encoding them as a term-by-document co-occurrence matrix, (i.e., a matrix where each row corresponds to a term and each column corresponds to a document, 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). LSA provides good results when applied to source code [22][16][21]. We use the LSA technique applied to models in the same way as [6].

The documents are text representations of model fragments. The text of the document corresponds to the names and values of the properties and methods of each model fragment (e.g. a model element of the class inductor will contain some properties related to its coil manufacturer and heat potential). The query is constructed from the terms that appear in the bug description. If the textual terms used for the model and the bug description differ too much, the LSA will not work. Therefore, the text from the documents (model fragments) and the text from the query (bug description) are homogenized by applying Natural Language Processing techniques (tokenizing [17], Parts-of-Speech Tagging [12], and Lemmatizing [20]) to eventually reduce this gap. The union of all the words extracted from the documents (model fragments) and from the query (bug description) are the terms (rows) used by our LSA fitness.

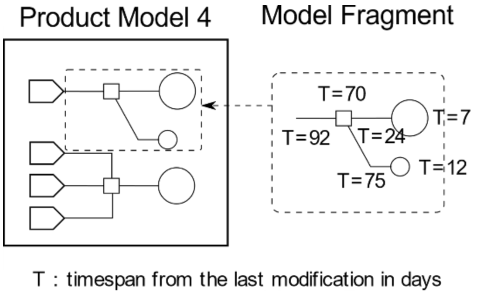
We normalize and decompose the matrix into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [14]. One vector that represents the latent semantics of the document is obtained for each model fragment and for the query. Finally, the similarities between the 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. For more details see [6].

3.2.2. Timespan Weightings

In this proposed fitness step, the modifications of the model over time are taken into consideration in order to extract the most relevant model for the target bug. In this section, we define the four timespan weighting functions used in our work.

These functions are based on the timespan between the last modification of a model element and the usage day. A recently modified model element (i.e. a short timespan) has a lower timespan value than another model element that was modified farther in the past. As a model fragment is composed by a set of model elements the timespan weighting of the model fragment depends on the timespan weightings of the model elements that compose it.

The timespan is based on the number of days and can therefore be very large when the model fragment was modified a long time ago. To normalize the timespans, mathematical solutions can be used. We used square roots because it has achieved good results in other works that use time differences [30]. The use of square root is more suitable and more effective for the proposed approach.



Approach	Timespan
BLiM-recent	7 days
BLiM-oldest	92 days
BLiM-mean	44.667 days
BLiM-sum	280 days

Fig. 3. Timespan of the modifications of the model elements of a fragment.

We devised four objective functions to capture the timespan weightings for the model fragments. Next, we define each of these functions:

The most recent model modifications (recent): this function expresses the concern of capturing primarily the model fragments with the model elements that have the lowest modification timespans. That is, model elements that have been recently modified. Then, the value of the model fragment will be the value of the most recently modified model element. In the example of Fig. 3, the value of the model fragment is 7 days, that means a square root of 2.646.

The oldest model modifications (oldest). This function expresses the concern of capturing primarily the model fragments with the model elements that have the highest modification timespans. That is, model elements that have not been modified for a long time, longer than most other elements. Then, the value of the model fragment will be the value of the model element less recently modified model element. In the example of Fig. 3, the value of the model fragment is 92 days, that means a square root of 9.592.

To avoid taking into account only the extremes (oldest and most recent modifications), we also define these two objective functions:

The mean of the timespan of the modified model elements (mean). This function expresses the concern of capturing primarily the model fragments with the model elements that have the lowest mean timespan. Then, the value of the model fragment will be the value of the mean of the timespan of the modified model elements. In the example of Fig. 3, the value of the model fragment is 46.667 days, that means a square root of 6.831.

The sum of the timespan of the modified model elements (sum). This function expresses the concern of capturing primarily the model fragments with the model elements that have the lowest timespan sum. Then, the value of the model fragment will be the value of the sum of the timespan of the modified model elements. In the example of Fig. 3, the value of the model fragment is 280 days, that means a square root of 16.733.

4. Evaluation: Bug Location in BSH

This section presents the evaluation of our approach: the experimental setup, a description of the case study where we applied the evaluation, the obtained results, the performed statistical analysis, and the threats to validity. To evaluate the approach, we applied it to an industrial case study from our partner, BSH: a leading manufacturer of home appliances in Europe.

4.1. Experimental Setup

The goal of this experiment is the evaluation of the different timespan weighting objective functions as fitness for our BLiM approach. In addition, we compare the BLiM approach with a baseline [7]. The baseline is the approach used in BSH for bug location. Although it was designed having a more general purpose in mind (Feature Location), it is the best they have for Bug Location in Models.

To evaluate our BLiM approach with the different objective functions (BLiM-recent, BLiM-oldest, BLiM-mean, and BLiM-sum) and the baseline approach, we run each of the approaches and obtain a ranking of model fragments that we can compare with an oracle in order to check accuracy. The inputs of the evaluation process, which are the product family, and bug reports, were provided by BSH.

The oracle is the ground truth, and is used to compare the results provided by the BLiM approach and the baseline. To prepare the oracle, BSH provided us with the bug reports that have occurred in the product models. These bug reports contain natural language bug descriptions and the approved bug realizations, which are a set of model fragments that realize the target bugs.

The baseline approach is a Single-Objective Evolutionary Algorithm (SOEA), whereas BLiM is Multi-Objective Evolutionary Algorithm (MOEA). The works in [13] shows that common MOEA measures such as hypervolume [31] are not necessarily suitable for comparing solutions by MOEAs (our BLiM approach) with solutions by SOEAs (baseline in this work). Therefore, in order to compare the baseline approach with BLiM, we first take the best solution of the baseline approach for its single-objective (the similarity with the bug description), and then we take the best solution of BLiM with regard to the objective of the baseline approach (the similarity with the bug description), as described in [13]. Finally, these solutions are compared to the bug realization of the oracle 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 both BLiM-X and the baseline) on a set of test data (the solutions) for which the true values are known (from the oracle). In our case, each solution outputted by the approaches is a model fragment composed of a subset of the model elements that are part of the product model (where the bug is being located). Since the granularity is at the level of model elements, each model element presence or absence is considered as a classification. The confusion matrix distinguishes between the predicted values and the real values classifying them into four categories:

True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle).

False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle).

True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle).

False Negative (FN): values that are predicted as false (in the solution) but are true in the real scenario (the oracle).

Then, some performance measurements are derived from the values in the confusion matrix. In particular, we create a report including three performance measurements (recall, precision, and F-measure), for each of the test cases for both BLiM-X and the baseline.

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:

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

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

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

4.2. Case Study

The case study where we applied our evaluation process is the Induction Hob Product Family of BSH (already presented in section 2 as the running example). The oracle is composed of 46 induction hob models, which are on average composed of more than 500 elements. BSH provided us with documentation of 37 bug reports and the approved bug realizations. For each of the 37 bugs, we created a test case that includes the set of product models where that bug was manifested and a bug description, both obtained from the documentation.

For this case study, we executed 30 independent runs for each of the 37 test cases for BLiM with all the different timespan weightings, and with the baseline (as suggested by [2]), i.e., 37 (bugs) x 5 (approaches) x 30 repetitions = 5550 independent runs.

4.3. Results

In this section, we present the results obtained by both BLiM (with the four fitness functions) and by the baseline approach, for the case study.

Table 1. Mean values and standard deviations for Recall, Precision and F-measure.

	BLiM-recent	**BLiM-oldest**	**BLiM-mean**	**BLiM-sum**	**Baseline**
Recall $\pm \sigma$	79.10±11.75	51.21±12.58	71.43±11.18	60.61±11.56	44.25±14.79
Precision $\pm \sigma$	73.26±9.44	27.99±7.74	64.91±9.57	45.69±12.45	29.04±9.47
F-measure $\pm \sigma$	76.07±8.34	36.20±7.46	68.02±7.54	52.10±8.26	35.07±9.00

Table 1 shows the mean values of recall, precision and F-measure of the graphs for both BLiM-X (with the four fitness functions) and the baseline, for the case study. BLiM-recent obtains the best results in recall and precision, providing an average value of 79.10% in recall and 73.26% in precision. The next best results are obtained by BLiM-mean, providing an average value of 71.43% in recall and 64.91% in precision. The third best values are obtained by BLiM-sum, providing an average value of 60.61% in recall and 45.69% in precision. BLiM-oldest obtains the worst value in precision, 27.99%, while the baseline approach obtains the worst value in recall, 44.25%. In terms of recall and precision, BLiM-recent outperforms the rest of the approaches.

From the results, we can see that there are some bugs (around 24% on average) that are not properly located by the approach. This happens because the fitness function that guides the search is not giving high fitness values to the model fragments realizing those bugs. This can happen due to differences between the language used in the bug descriptions and the product models, or in cases where there are few differences in the modification timespan among the different model fragments.

4.4. Statistical Analysis

To properly compare our BLiM approach (with the four fitness functions) and the baseline approach, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [2]. The goals of our statistical analysis are: (1) to provide formal and quantitative evidence (statistical significance) that BLiM-recent 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).

4.4.1. Statistical significance

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

The statistical tests provide a probability value, $p-value$. The $p-value$ obtains values between 0 and 1. The lower the $p-value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p-value$ under 0.05 is statistically significant [2], and so the hypothesis H_0 can be considered false.

The test that we must follow depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data; however, the Quade test shows that it is the most powerful when working with real data [8]. In addition, according to Conover [3], the Quade test is the one that has shown the best results when the number of algorithms is low (no more than 4 or 5 algorithms).

The $p-value$ obtained in the test are $\ll 2x10^{-16}$ for recall and precision, the statistics value obtained are 32.628 and 62.196 for recall and precision respectively. Since the $p-value$ are smaller than 0.05 for recall and precision, we reject the null hypothesis. Consequently, we can state that there exist differences among the algorithms (BLiM-recent, BLiM-oldest, BLiM-mean, BLiM-sum, and the baseline) for the performance indicators of recall and precision.

However, with the Quade test, we cannot answer the following question: Which of the algorithms gives the best performance? In this case, the performance of each algorithm should be individually compared against all other alternatives. In order to do this, we perform an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each algorithm, determining whether statistically significant differences exist among the results of a specific pair of algorithms.

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

	Holm's	**$\hat{A}_{12}$**

	Recall	Precision	Recall	Precision
recent vs. oldest	$4x10^{-12}$	$\ll 2x10^{-16}$	0.9437546	1
recent vs. mean	0.0434	0.019	0.6775018	0.7319211
recent vs. sum	$1.1x10^{-6}$	$7.6x10^{-10}$	0.8699781	0.9466764
recent vs. baseline	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	0.9656684	1
oldest vs. mean	$5.8x10^{-7}$	$\ll 2x10^{-16}$	0.1278305	0
oldest vs. sum	0.0416	$1.4x10^{-6}$	0.2885318	0.1022644
oldest vs. baseline	0.0527	0.939	0.6449963	0.4598247
mean vs. sum	0.0078	$9.1x10^{-5}$	0.7465303	0.8663258
mean vs. baseline	$2.7x10^{-11}$	$\ll 2x10^{-16}$	0.9181885	1
sum vs. baseline	$8.7x10^{-5}$	$1.3x10^{-6}$	0.7991234	0.8363769

Table 2 shows the $p - value$ of Holm's post hoc analysis for the case study and the performance indicators for the five algorithms (BLiM-recent, BLiM-oldest, BLiM-mean, BLiM-sum, and the baseline). The majority of the $p - value$ shown in this table are smaller than their corresponding significance threshold value (0.05), indicating that the differences of performance between the algorithms are significant. However, when comparing BLiM-oldest and the baseline (seventh row), the values are greater than the threshold, indicating that the differences between those algorithms could be due to the stochastic nature of the algorithms and are not significant.

4.4.2. Effect size

When comparing algorithms with a large enough number of runs, statistically significant differences can be obtained even if they are so small as to be of no practical value [2]. Then it is important to assess if an algorithm is statistically better than another and to assess the magnitude of the improvement. Effect size measures are needed to analyze this.

For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [28][9]. $\hat{A}_{12}$ measures the probability that running one algorithm yields higher values than running another algorithm. If the two algorithms are equivalent, then $\hat{A}_{12}$ 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results in 70% of the runs with the first pair of algorithms that have been compared, and $\hat{A}_{12} = 0{,}3$ means that we would obtain better results in 70% of the runs with the second pair of algorithms that have been compared. Thus, we have an $\hat{A}_{12}$ value for every pair of algorithms.

Table 2 shows the values of the size effect statistics. In general, the largest differences were obtained between BLiM-recent and the baseline, where BLiM-recent achieves better recall than the baseline 96% of the times and better precision almost all the times. When comparing BLiM-recent and BLiM-mean the differences are not so big, with BLiM-recent outperforming BLiM-mean in recall 67% of the times and in precision 73% of the times.

BLiM-recent obtained the best performance results among the five evaluated approaches (see Table 1). The performed statistical analysis indicated that BLiM-recent outperforms the rest of the approaches in terms of recall and precision (around 70% of the times when compared to BLiM-mean, 90% of the times when compared to BLiM-sum and almost all the times when compared to BLiM-oldest and the baseline). Overall, these results confirm that the use of BLiM-recent against the baseline approach has an actual impact.

4.4.3. Threats to Validity

In this section, we present some of the threats to the validity of our work. We follow the guidelines suggested by De Oliveira et. al [19] to identify those applicable to this work.

Conclusion validity threats: To address the *not accounting for random variation* threat, we considered 30 independent runs for each bug with each algorithm. As we used the approach that BSH uses for bug location as a comparison baseline, the *lack of a meaningful comparison baseline* threat is addressed. In this paper we employed standard statistical analysis following accepted guidelines [2] to avoid the *lack of a formal hypothesis and statistical tests* threat. The fourth threat is the *lack of a good descriptive analysis*. For avoid the *lack of a formal hypothesis* *and statistical tests* threat, we have used the precision, recall, and F-measure measurements to analyze the confusion matrix obtained from the experiments; however, other measurements could be applied.

Internal validity threats: To address the *poor parameter settings* threat, we used standard values for the algorithms. As suggested by Arcuri and Fraser [2], default values are good enough to measure the performance of location techniques in the context of testing. Nevertheless, we plan to evaluate all the parameters of our algorithm in a future work. As we have evaluated our work in an industrial case study the *lack of real problem instances* threat is addressed

Construct validity threats: To address the *lack of assessing the validity of cost measures* threat, we performed a fair comparison among BLiM-X and the baseline by generating the same number of model fragments and using the same number of fitness evaluations.

External validity threats: The *lack of a clear object selection strategy* threat is addressed by using an industrial case study, BSH. Our instances are collected from real world problems.

5. Related Work

Saha et al. [24] presented BLUiR, which uses a baseline "TF.IDF model". They believe that code constructs improve the accuracy of bug localization. They syntactically parse the source code into four document fields: class, method, variable, and comment. The summary and the description of a bug report are considered as two query fields. Textual similarities are computed for each of the eight-document field-query field pairs and then summed up into an overall ranking measure. Kim et al. [5] propose both a one-phase and a two-phase prediction model to recommend files to fix. In the one-phase model, they create features from textual information and metadata of bug reports, apply Naïve Bayes to train the model using previously fixed files as classification labels, and then use the trained model to assign multiple source files to a bug report. In the two-phase model, they first apply their one-phase model to classify a new bug report as either "predictable" or "deficient", and then make predictions only for "predictable" reports. Unlike us, all of these approaches do not take into account the modification timespan of the retrieved source locations. Furthermore, these approaches target code while our approach targets models to locate the bug realizations.

Zamani et al [29] proposed an approach that included weighting and ranking the source code locations based on both the textual similarity with a change request and the use of the time metadata. This approach gives better results than IR techniques. However, their approach is applied at the source code level, while we use a Multi-Objective Evolutionary Algorithm to address the location of bugs in models.

In addition, other approaches use genetic algorithms to locate features in models, Font et al. [6, 7] propose two approaches to locate features in a model. However, these works do not take into account the modification timespan of the model elements. Our work, in contrast, is focused on searching bug realizations, hence, the timespan weighting is an important piece of the approach in order to obtain accurate results.

6. Conclusion

Bug Location is a significant maintenance activity. In this paper, we have proposed four approaches for bug location in models (BLiM-recent, BLiM-oldest, BLiM-mean, BLiM-sum) and compared them with a baseline. Our BLiM-X approaches, in order to guide our bug location evolutionary algorithm, consider: (1) the similitude to the bug description, and (2) the modification timespan weightings of the models.

We evaluate which approach produces better results in terms of precision, recall and F-measure. To do so, we applied the five approaches in an industrial domain, BSH, that has a model based product family (firmware of Induction Hobs). We report our evaluation, including: experimental setup, results, statistical analysis, and threats to validity.

The results show that the modification timespan weightings of models pay off for bug location. In particular, using the most recent modification timespan of a model element (BLiM-recent) provides the best results in our study. Results also show that our approach can be applied in real world environments. The statistical analysis of the results provides evidence of their significance.

Acknowledgements

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

References

1. Antoniol, G., Gueheneuc, Y.-G.: Feature Identification: An Epidemiological Metaphor. IEEE Trans. Softw. Eng. 32 (9), 627–641 (2006)
2. Arcuri, A., Fraser, G.: Parameter tuning or default values? An empirical investigation in search-based software engineering. Empir. Softw. Eng. 18 (3), 594–623 (2013)
3. Conover, W.J.: Practical Nonparametric Statistics, 3rd Edition. Wiley (1999)
4. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature Location in Source Code: A Taxonomy and Survey. In: Journal of Software Maintenance and Evolution: Research and Practice. (2011)
5. Dongsun Kim, Yida Tao, Sunghun Kim, Zeller, A.: Where Should We Fix This Bug? A Two-Phase Recommendation Model. IEEE Trans. Softw. Eng. 39 (11), 1597–1610 (2013)
6. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: Feature Location in model-based Software Product Lines through a Genetic Algorithm. In: 15th International Conference on Software Reuse. , Limassol, Cyprus (2016)
7. Font, J., Arcega, L., Haugen, Ø., Cetina, C.: 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. pp. 272–282. ACM, New York, NY, USA (2016)
8. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Inf. Sci. (Ny). 180 (10), 2044–2064 (2010)
9. Grissom, R.J., Kim, J.J.: "Effect sizes for research: A broad practical approach. Mahwah, NJ: Earlbaum (2005)
10. Hassan, A.E., Holt, R.C.: The top ten list: dynamic fault prediction. In: 21st IEEE International Conference on Software Maintenance (ICSM'05). pp. 263–272. (2005)
11. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding Standardized Variability to Domain Specific Languages. In: Proceedings of the 2008 12th International Software Product Line Conference. pp. 139–148. IEEE Computer Society, Washington, DC, USA (2008)
12. Hulth, A.: Improved Automatic Keyword Extraction Given More Linguistic Knowledge. In: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing. pp. 216–223. Association for Computational Linguistics, Stroudsburg, PA, USA (2003)
13. Ishibuchi, H., Nojima, Y., Doi, T.: Comparison between Single-Objective and Multi-Objective Genetic Algorithms: Performance Comparison and Performance Measures. In: 2006 IEEE International Conference on Evolutionary Computation. pp. 1143–1150. (2006)
14. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Process. 25 (2–3), 259–284 (1998)
15. Lehman, M.M., Ramil, J.F., Kahen, G.: A paradigm for the behavioural modelling of

software processes using system dynamics. Citeseer (2001)

16. Liu, D., Marcus, A., Poshyvanyk, D., Rajlich, V.: 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. pp. 234–243. ACM, New York, NY, USA (2007)
17. Manning, C.D., Schütze, H.: Foundations of Natural Language Processing. Reading. 678 (2000)
18. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.I.: An information retrieval approach to concept location in source code. In: Proceedings of the 11th Working Conference on Reverse Engineering. pp. 214–223. (2004)
19. de Oliveira Barros, M., Dias-Neto, A.C.: 0006/2011-Threats to Validity in Search-based Software Engineering Empirical Studies. RelaTe-DIA. 5 (1), (2011)
20. Plisson, J., Lavrac, N., Mladenic, D.: A rule based approach to word lemmatization. In: Proceedings of the 7th International Multi- Conference Information Society IS 2004. pp. 83–86. (2004)
21. Poshyvanyk, D., Gueheneuc, Y.-G., Marcus, A., Antoniol, G., Rajlich, V.: Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. IEEE Trans. Softw. Eng. 33 (6), 420–432 (2007)
22. Revelle, M., Dit, B., Poshyvanyk, D.: Using Data Fusion and Web Mining to Support Feature Location in Software. In: IEEE 18th International Conference on Program Comprehension (ICPC). pp. 14–23. (2010)
23. Rubin, J., Chechik, M.: A Survey of Feature Location Techniques. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., and Bettin, J. (eds.) Domain Engineering. pp. 29–58. Springer Berlin Heidelberg (2013)
24. Saha, R.K., Lease, M., Khurshid, S., Perry, D.E.: Improving bug localization using structured information retrieval. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 345–355. (2013)
25. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA (1986)
26. Sisman, B., Kak, A.C.: Incorporating version histories in Information Retrieval based bug localization. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR). pp. 50–59. (2012)
27. Svendsen, A., Zhang, X., Lind-Tviberg, R., Fleurey, F., Haugen, Ø., Møller-Pedersen, B., Olsen, G.K.: Developing a Software Product Line for Train Control: A Case Study of CVL. In: Proceedings of the 14th International Conference on Software Product Lines: Going Beyond. pp. 106–120. Springer-Verlag, Berlin, Heidelberg (2010)
28. Vargha, A., Delaney, H.D.: A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. J. Educ. Behav. Stat. 25 (2), 101–132 (2000)
29. Zamani, S., Lee, S.P., Shokripour, R., Anvik, J.: A noun-based approach to feature location using time-aware term-weighting. Inf. Softw. Technol. 56 (8), 991–1011 (2014)
30. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A.: Mining Version Histories to Guide Software Changes. In: Proceedings of the 26th International Conference on Software Engineering. pp. 563–572. IEEE Computer Society, Washington, DC, USA (2004)
31. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms --- A comparative case study. In: Eiben, A.E., Bäck, T., Schoenauer, M., and Schwefel, H.-P. (eds.) Parallel Problem Solving from Nature --- PPSN V: 5th International Conference Amsterdam, The Netherlands September 27--30, 1998 Proceedings. pp. 292–301. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)