Studying the Influence and Distribution of the Human Effort in a Hybrid Fitness Function for Search-Based Model-Driven Engineering

Rodrigo Casamayor, Carlos Cetina, Óscar Pastor, *Member, IEEE*, and Francisca Pérez

***Abstract*—Search-Based Software Engineering (SBSE) offers solutions that efficiently explore large complex problem spaces. To obtain more favorable solutions, human participation in the search process is needed. However, humans cannot handle the same number of solutions as an algorithm. We propose the first hybrid fitness function that combines human effort with human simulations. Human effort refers to human participation for providing evaluations of candidate solutions during the search process, whereas human simulations refer to recreations of a scenario in a specific situation for automatically obtaining the evaluation of candidate solutions. We also propose three variants for the hybrid fitness function that vary in the distribution of human effort in order to study whether the variants influence the performance in terms of solution quality. Specifically, we leverage our hybrid fitness function to locate bugs in software models for the video games of game software engineering. Video games are a fertile domain for these hybrid functions because simulated players are naturally developed as part of the video games (e.g., bots in First-Person Shooters). Our evaluation is at the scale of industrial settings with a commercial video game (Play Station 4 and Steam) and 29 professional video game developers. Hybridizing the fitness function outperforms the results of the best baseline by 33.46% in F-measure. A focus group confirms the acceptance of the hybrid fitness function. Hybridizing the fitness function significantly improves the bug localization process by reducing the amount of tedious manual work and by minimizing the number of bugs that go unnoticed. Furthermore, the variant that obtains the best results is a counter-intuitive result that was under the radar of the interactive SBSE community. These results can help not only video game developers to locate bugs, but they can also inspire SBSE researchers to bring hybrid fitness functions to other software engineering tasks.**

***Index Terms*—Interactive SBSE, search-based software engineering, bug localization, video games, model-driven engineering.**

I. Introduction

SEARCH-Based Software Engineering (SBSE) addresses problems throughout the Software Engineering lifecycle (from requirements to maintenance) using search-based algorithms. Increasingly, the community has been paying more attention to SBSE [1] since it offers automated or semi-automated solutions that efficiently explore large complex problem spaces. For example, SBSE can be used to locate model fragments from an input query in a family of software products that has been developed and maintained by a company over years [2]. Only three key ingredients are needed to apply SBSE: 1) a representation (encoding) of the problem (e.g., a bit string); 2) the definition of the set of operations (e.g., mutation); and 3) the definition of the fitness function (e.g., similarity to the input query). Then, candidate solutions (which are encoded following the representation chosen) are evolved (by applying the operations) and are assessed (by the fitness function) in an iterative process until a stop condition is met (e.g., a time slot).

Although SBSE reformulates software engineering problems as search problems, some contexts require the human's subjective evaluation in the search process in order to obtain the most favorable solutions that alleviate some limitations of SBSE techniques (e.g., vocabulary mismatch and tacit knowledge). This refers to interactive SBSE (iSBSE) [3]. For example, Wang et al. [4] suggest that there are complex approaches that are partially successful on multiple fault problems that need more human intervention. Marculescu et al. [5] involve a human in the fitness function when the optimization goal depends on ”human preference, intuition, emotion, and psychological aspects”. Feldt [6] emphasizes the importance of collaboration between a human and a system in identifying and comprehending a bug's root cause. Most previous iSBSE works [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22] interleave the human's subjective evaluation with the evaluation that is performed by the algorithm until the stop condition is met. Although other works use the human's subjective evaluation differently (before [23], [24], [25], [26] or after [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39] the algorithm), none of these iSBSE works compare whether

Manuscript received 1 March 2023; revised 27 October 2023; accepted 29 October 2023. Date of publication 10 November 2023; date of current version 12 December 2023. This work was supported in part by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project VARIATIVA under Grant PID2021-128695OB-I00 and in part by the Gobierno de Aragón (Spain) (Research Group S05 20D). Recommended for acceptance by T. Menzies. *(Corresponding author: Rodrigo Casamayor.)*

Rodrigo Casamayor and Francisca Pérez are with the SVIT Research Group, Universidad San Jorge, 50830 Zaragoza, Spain (e-mail: rcasamayor@usj.es; ccetina@usj.es; mfperez@usj.es).

Carlos Cetina is with the SVIT Research Group, Universidad San Jorge, 50830 Zaragoza, Spain, and also with the University College London, WC1E 6BT London, U.K.

Óscar Pastor is with the Research Center on Software Production Methods (PROS), Universitat Politècnica de València, 46022 Valencia, Spain (e-mail: opastor@dsic.upv.es).

Digital Object Identifier 10.1109/TSE.2023.3329730

a different distribution of the human evaluations improves the quality of the solutions.

For the first time, our previous work [17] completely replaced the fitness function with a human for Search-Based Model-Driven Engineering (SBMDE), specifically, for locating features in software models. Software models are a more favorable context for humans than code because of their higher abstraction. Although involving a human as fitness function (HaFF) improves the results, human participation must be limited in order to prevent fatigue [40]. All in all, humans are not immune to fatigue, so logically they cannot handle and process the same number of evaluation requests as an algorithm. According to Takagi's [40] recommendations for avoiding human fatigue, both the size of the population to be evaluated and the number of iterations should be 10 or 20.

For more demanding problems (e.g., large-scale software systems) where human fatigue may appear due to the high number of human evaluations that are needed to locate a solution, it is necessary to combine the effort of an algorithm that automatically obtains the best solution possible with the human. Hybridizing the fitness function by exploiting both the algorithm and the human as fitness function in SBMDE can enrich the solution quality. Hence, in this work, we propose a hybrid fitness function that is the first to combine human effort with human simulations. On the one hand, human effort refers to human participation for providing manual evaluations of candidate solutions during the search process. On the other hand, human simulations refer to recreations of a scenario in a specific situation for automatically obtaining the evaluation of candidate solutions. Building a simulated human may be a bigger challenge than the task at hand. However, in the domain of video games that Game Software Engineering (GSE) tackles [41], there are human simulations that have been built as part of the development of the video game. Some examples of human simulations in video games are: the rival drivers in a racing game, the bots in a First-Person Shooter (FPS), or the generals of the enemy troops in a Real-Time Strategy (RTS).

In this work, we study the influence of hybridizing the fitness function in terms of solution quality in the context of Bug Localization (BL) in software models for GSE. Moreover, we propose three variants of the hybrid fitness function in order to study the influence in the quality of the solution of different ways of distributing the human effort during BL: Variant 1 interleaves the assessment between the human and the algorithm; Variant 2 relies on the human assessment first, and then the assessment of the algorithm; and Variant 3 relies on the assessment the algorithm first, and then the human assessment.

In the evaluation, 29 professional video game developers were involved, acting as the human component of the fitness function with the objective of locating 29 bugs in Kromaia, which is a commercial video game about flying and shooting with a spaceship in a three-dimensional space[1]. Kromaia was released on PC, PlayStation and translated to eight different languages.

To assess the performance of our hybrid fitness function in terms of solution quality, we apply metrics (recall, precision, and F-measure) that have been widely accepted by the software engineering community in the domain of evolutionary algorithms [42]. In order to put the performance of the variants into perspective, we set two baselines that include: only using the human as fitness function as proposed in [17], and only using an algorithmic fitness function as proposed in [43]. In order to compare the results of the variants with the baselines, we perform a statistical analysis (following the guidelines by Arcuri and Briand [44]) in order to provide quantitative evidence of the impact of the results and to show that this impact is significant.

The results show that all of the variants of our hybrid fitness function significantly outperformed the baselines in F-measure. Specifically, Variant 3 (the algorithm assessments first and the human afterwards) was the one that yielded the best results, improving the results of the best baseline (only using the human as fitness function) by 40.3% in recall, 26.46% in precision, and 33.46% in F-measure. All of the comparisons show significant differences except when the recall of Variant 1 (interleaving the assessment between the human and the algorithm) and Variant 2 (the human assessment first and the algorithm afterwards) is compared to the baseline that only uses the algorithm as fitness function. The significant improvement in precision and F-measure in all of the comparisons comes from the influence of the human on the unattended algorithm. According to the magnitude scales of the Cliff Delta values [45], the magnitude of improvement using Variant 3 instead of the baselines can be considered large.

To the best of our knowledge, this is the first SBMDE work that empowers humans with a hybrid fitness function that combines human assessment with simulations during BL in GSE and which obtains more favorable solutions than the baselines at an industrial scale. Specifically, we claim that:

- Our hybrid fitness function significantly improves the results of BL compared to the baselines. Furthermore, we created a focus group that confirms the acceptance of using a hybrid function in the context of BL for GSE. Our results can also motivate other researchers to use a hybrid fitness function to benefit other software maintenance activities.
- The variant of the hybrid fitness function that obtains the best results distributes all of the effort of the algorithm at the beginning and the human afterwards (Variant 3). This result is counter-intuitive from an initial survey that we made. The group of video game developers responded that Variant 1 would obtain the best results (interleaving simulations automatically produced by the evolutionary algorithm with human evaluations).
- Previous works that have involved the human in the fitness function (e.g., the human refines the algorithm's solution [21]) interleave the human participation and the algorithm execution. These works do not consider other approaches to distribute human effort. Our work reveals that interleaving is not the approach that achieves the best results. We acknowledge that our results are obtained in a different context of simulation-human-based hybrid fitness function, but our results might motivate other researchers to reconsider the decision of using interleaving as the default approach.

[1] See the official Playstation trailer to learn more about Kromaia: https://youtu.be/EhsejJBp8Go

- The current shortage of bug-localization approaches in video games results in longer development times, delayed deadlines, and postponed release dates. Hybridizing the fitness function significantly improves the BL process by reducing the amount of tedious manual work and minimizing the number of bugs that go unnoticed.

The remainder of this paper is organized as follows. Section II introduces bug localization in the GSE domain. Section III reviews the related work. Section IV describes the variants that we propose for hybridizing and distributing human effort as fitness function. Section V presents the evaluation. The results are reported in Section VI and discussed in Section VII. Section VIII describes the threats to validity. Finally, Section IX concludes the paper.

II. Background

This section introduces how SBMDE is used to locate bugs in the software models of a commercial video game by leveraging game simulations. In this context, we also motivate the need for an approach that involves a human in the evaluation of candidate solutions. This section also introduces iSBSE.

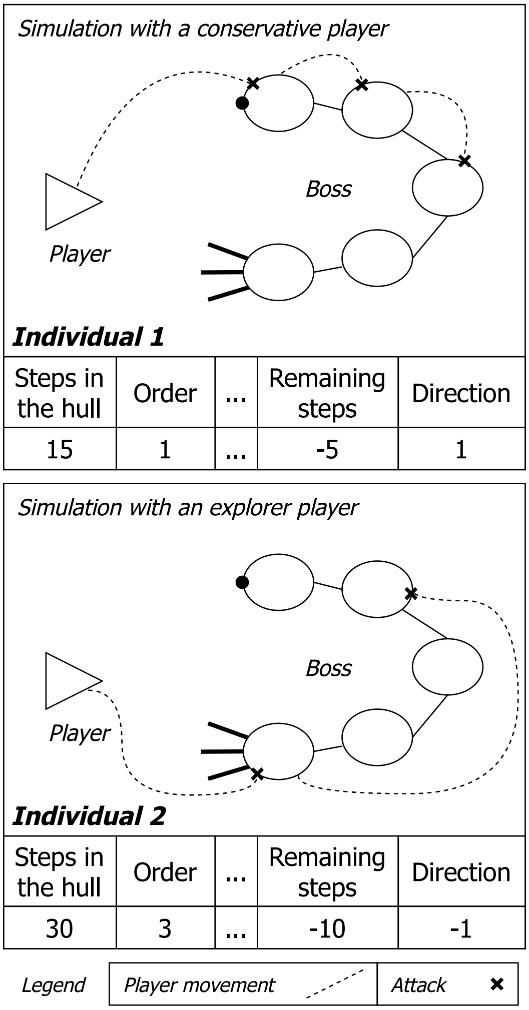
A. SBMDE for Locating Bugs in a Commercial Video Game

Our previous work [43] uses SBMDE to find the model fragments that are the source of bugs in a commercial video game (Kromaia). The content of Kromaia is specified using the Shooter Definition Model Language (SDML), which is a Domain Specific Language (DSL) model for the video game domain [46]. This DSL adheres to the fundamental principles of MDE using models for Software Engineering. The models are built with SDML and interpreted at runtime.

SDML defines aspects that are included in video game entities such as bosses, which must be defeated in order to complete a level. The definition of a boss includes: the anatomical structure (including which parts are used in it, their physical characteristics, and how they are connected to one another); the quantity and distribution of vulnerable parts, weapons, and defenses in the structure/body of the character; and the movement behaviors associated to the whole body or its parts. This modeling language includes concepts such as hulls, links, weak points, weapons, and AI components.

Fig. 1 shows a simplified example of the graphical representation of a boss and a player. For further information, the following URL contains examples of the models, the metamodel, and an online visualizer that displays the models as they would appear in the Kromaia video game: https://svit.usj.es/tse23/bl-in-mgse.

To locate the bugs in Kromaia, our previous work [43] uses game simulations. The simulations mimic a duel between a boss and an algorithm that can behave like a human player (i.e., the simulated player). During a simulation, the boss acts in accordance with the anatomy, behavior, and attack/defense balance that are included in its model, trying to defeat the simulated player, whereas the simulated player confronts the boss in order to destroy the available weak points. Both the boss and the simulated player try to win the fight, avoid draw/tie games, and make sure there is a winner in the simulation.



Steps in the hull	Order	...	Remaining steps	Direction
15	1	...	-5	1

Steps in the hull	Order	...	Remaining steps	Direction
30	3	...	-10	-1

Fig. 1. Examples of simulations between a player and boss.

Fig. 1 shows two examples of simulations. Each simulation emulates the behavior of a player when the battle with the boss occurs. The algorithm can fight a boss by applying different strategies. The fighting strategy can therefore be defined by parametrizing the algorithm. Thus, different player profiles can be formed. For example, the parameters can define how many steps the simulated player takes in each hull of the boss, the order in which the hulls are visited following different patterns (one by one, visit one skip one, visit one skip three...), if the player requires all of the remaining steps in the hull when he/she is attacked by it, or the direction used to visit the hulls of the boss (forward or backward).

Each example in Fig. 1 corresponds to different fighting strategies applied to a simulation. In both cases, the triangle corresponds to the simulated player, the set of connected circles and lines correspond to the boss, the dashed and dotted lines correspond to the path that follows the simulated player in his/her strategy to defeat the boss, and the crosses correspond to the attacks that the simulated player performs to the hulls. The upper example of Fig. 1 shows the simulation of a conservative player in which the player attacks each hull one at a time. The lower example of the figure shows the simulation of an explorer player in which the player attacks the last hull of the boss, skips two hulls, and then attacks the following hulls up to the head of the boss.

Fig. 1 shows a set of parameters applied to a simulation for each example. Each set of parameters defines a candidate solution (individual). Each individual is encoded as a vector representation whose size corresponds to the number of parameters (dimensions) in the vector. The simulation parameters were provided by the developers based on the analysis of battles between real players and bosses. Furthermore, a scenario is defined as the union between the simulation parameters (i.e., duel settings), the agent parameters (i.e., the parameters that define the behavior of a player in the simulation), and the software model (e.g., the model of the boss) to be analyzed for potential bug localization. The fitness function of our previous SBMDE work [43] rewards simulations that are the farthest from what is expected (i.e., the further from what is expected, the more relevant when it comes to locating a bug).

To obtain more favorable solutions, the human should be involved in the search process as previous works have detected [4], [5], [6], [17]. However, human participation must be limited to prevent fatigue from demanding problems where the humans cannot handle the same number of candidate solutions as an algorithm [17], [40]. For these demanding problems, we propose to study the influence and distribution of human effort in a hybrid fitness function during the location of model fragments that are sources of bugs.

B. Interactive SBSE

Interactive SBSE (iSBSE) has been formalized as an emergent subarea within SBSE that promotes active human effort by providing intermediate results for inspection by the humans. Human feedback is later integrated into either the problem formulation or the search process so that the algorithm progressively adapts the search to the human's preferences [18], [40]. A recent survey on iSBSE [3] acknowledges that any attempt to involve the human in the search process with the aim of adapting the results to the human's preferences can be viewed as iSBSE.

The use of interactivity during the search process has shown more favorable solutions than fully automated solutions since some limitations of SBSE techniques (e.g., vocabulary mismatch and tacit knowledge) are alleviated [3], [4], [5], [6], [17].

Nevertheless, iSBSE also poses new challenges like dealing with the inherent human fatigue. Table I shows the strategies for addressing human fatigue of the 33 iSBSE works, which are also compared in the next section (related work). Most of the iSBSE works (81.82%) do not report strategies for addressing human fatigue. Specifically, 57.58% of the iSBSE works do not explicitly mention fatigue ("-" in Column 2 of the table), whereas 24.24% of the works explicitly mention that they do not limit the time or iterations for human evaluations ("No limit" in Column 2 of the table). The rest of the iSBSE works (18.18%) limit the human participation with a set of interaction-related parameters to adapt the human participation to their needs ("Parameter Limit: *parameter*" in Column 2 of the table). These parameters limit the number of interactions, evaluations, or the time that the human can spend evaluating. The previous parameters limit the human participation to avoid fatigue.

TABLE I
STRATEGIES FOR ADDRESSING HUMAN FATIGUE IN ISBSE RELATED WORK

	Strategy for Addressing Fatigue
Ghannem et al. [7]	–
Amal et al. [8]	–
Lin et al. [27]	–
Yue et al. [23]	–
Van Rooijen and Hamann [28]	–
Lu et al. [24]	–
Debreceni et al. [29]	–
Batot and Sahraoui [30]	No limit
Fleck et al. [31]	–
Martínez et al. [9]	Parameter Limit: interactions
Gómez-Abajo et al. [32]	–
Calinescu et al. [33]	–
Araujo et al. [10]	No limit
Kessentini et al. [11]	No limit
Martínez et al. [12]	–
Marculescu et al. [13]	–
Kolchin [34]	–
Filho et al. [25]	Parameter Limit: evaluations
Bindewald et al. [14]	Parameter Limit: interactions
Procter et al. [35]	–
Le Calvar et al. [36]	–
Zubcoff et al. [37]	–
Alkhazi et al. [38]	–
Alkhazi et al. [39]	No limit
Silva et al. [26]	–
Alizadeh et al. [15]	No limit
Kessentini et al. [16]	No limit
Pérez et al. [17]	Parameter Limit: interactions and evaluations
Ramírez et al. [18]	Parameter Limit: time
Kuviatkovski et al. [19]	No limit
Delgado-Pérez et al. [20]	Parameter Limit: interactions
Kessentini et al. [21]	No limit
Freire et al. [22]	–

As in the previous works, we also limit the number of human evaluations and iterations in this work to address human fatigue since humans cannot evaluate the same solutions as an algorithm. To set this limit, we follow Takagi's [40] recommendations, which are that both the size of the population to be evaluated and the number of iterations should be 10 or 20. Therefore, we address fatigue as in previous works (limiting the human participation), but, in this work, we focus on studying the influence of different ways of distributing the human effort during BL on the quality of the solution.

III. Related Work

Table II shows the related work, which is organized in two main parts in order to take into account the topics that are covered in the paper (BL and human effort in a hybrid fitness function). Also, the lower part of the table includes a row to compare the related work with our work. The columns of the table show: the related work (Column 1); if the work includes human interaction before, during, or after of the search process (Column 2); if the human participation is done during the search

TABLE II
RELATED WORK IN BUG LOCALIZATION AND iSBSE

		In the fitness function			
	Human interaction	Human assessment	Hybrid: simulation and human	Studies human effort distribution	Industrial scale
Burgueño et al. [47]	✗	✗	✗	✗	✗
Iftikhar et al. [48]	✗	✗	✗	✗	✓
Sánchez-Cuadrado et al. [49]	✗	✗	✗	✗	✗
Sánchez-Cuadrado et al. [50]	✗	✗	✗	✗	✗
Troya et al. [51]	✓	✗	✗	✗	✗
Ariyurek et al. [52]	✗	✗	✗	✗	✗
Zheng et al. [53]	✗	✗	✗	✗	✓
Ariyurek et al. [54]	✗	✗	✗	✗	✗
Cheng et al. [55]	✗	✗	✗	✗	✗
Zhang et al. [56]	✗	✗	✗	✗	✗
Arcega et al. [57]	✓	✗	✗	✗	✓
Ferdous et al. [58]	✗	✗	✗	✗	✗
Quach et al. [59]	✗	✗	✗	✗	✗
Casamayor et al. [43]	✗	✗	✗	✗	✓
Ciborowska et al. [60]	✗	✗	✗	✗	✓
Khanfir [61]	✗	✗	✗	✗	✓
Liang et al. [62]	✗	✗	✗	✗	✓
Tufano et al. [63]	✗	✗	✗	✗	✓
Ghannem et al. [7]	✓	✓	✗	✗	✗
Amal et al. [8]	✓	✗	✗	✗	✗
Lin et al. [27]	✓	✗	✗	✗	✗
Yue et al. [23]	✓	✗	✗	✗	✗
Van Rooijen and Hamann [28]	✓	✗	✗	✗	✗
Lu et al. [24]	✓	✗	✗	✗	✗
Debreceni et al. [29]	✓	✗	✗	✗	✓
Batot and Sahraoui [30]	✓	✗	✗	✗	✗
Fleck et al. [31]	✓	✗	✗	✗	✗
Martínez et al. [9]	✓	✓	✗	✗	✗
Gómez-Abajo et al. [32]	✓	✗	✗	✗	✗
Calinescu et al. [33]	✓	✗	✗	✗	✗
Araujo et al. [10]	✓	✓	✗	✗	✓
Kessentini et al. [11]	✓	✓	✗	✗	✗
Martínez et al. [12]	✓	✓	✗	✗	✗
Marculescu et al. [13]	✓	✓	✗	✗	✓
Kolchin [34]	✓	✗	✗	✗	✗
Filho et al. [25]	✓	✗	✗	✗	✗
Bindewald et al. [14]	✓	✓	✗	✗	✗
Procter et al. [35]	✓	✗	✗	✗	✗
Le Calvar et al. [36]	✓	✗	✗	✗	✗
Zubcoff et al. [37]	✓	✗	✗	✗	✗
Alkhazi et al. [38]	✓	✗	✗	✗	✗
Alkhazi et al. [39]	✓	✗	✗	✗	✗
Silva et al. [26]	✓	✗	✗	✗	✗
Alizadeh et al. [15]	✓	✓	✗	✗	✓
Kessentini et al. [16]	✓	✓	✗	✗	✗
Pérez et al. [17]	✓	✓	✗	✗	✓
Ramírez et al. [18]	✓	✗	✗	✗	✗
Kuviatkovski et al. [19]	✓	✗	✗	✗	✗
Delgado-Pérez et al. [20]	✓	✗	✗	✗	✓
Kessentini et al. [21]	✓	✓	✗	✗	✗
Freire et al. [22]	✓	✓	✗	✗	✗
Our work	✓	✓	✓	✓	✓

process in the fitness function by assessing candidate solutions (Column 3); if the fitness function is a hybrid combining a simulation and the effort of a human (Column 4); whether or not the distribution of the human effort in the fitness function is studied (Column 5); and if the evaluation performed explicitly mentions that industry is involved (Column 6). In each cell of the table, we use either a check mark (to indicate that the work explicitly addresses what is mentioned in the column) or a cross mark otherwise.

The upper part of Table II includes 18 BL works which mainly cover: BL in games, BL in models, and BL in games that use models. Using the query presented in [43], we collected new papers up to January 2023 and identified those already covered to maintain consistency. For instance, Troya et al. [51] locate faulty rules in model transformations by applying Spectrum-Based Fault Localization (SBFL), where the human picks one assertion from the ranking to locate and fix the faulty rule that made the assertion fail. Arcega et al. [57] evaluate different model-based BL approaches in order to mitigate the effect of starting the localization in the wrong place. Software engineers are enabled to modify the solution obtained (e.g., by adding or removing model elements to a model fragment that the BL approach has obtained). Our previous work in BL in software models of video games, Casamayor et al. [43], uses an evolutionary algorithm where the candidate solutions are automatically assessed by a fitness function that collects information about the simulations. Only two BL works [51], [57] include human interaction, but neither of them consider the human in the assessment of candidate solutions in the fitness function, as Column 3 of Table II shows. Therefore, none of these BL works have a hybrid fitness function that combines a simulation and the effort of a human or the study of how human effort should be distributed to obtain the best quality solution possible.

The middle part of Table II shows the works of two recent surveys that include SBMDE works from 1998 to 2016 [64], and iSBSE works from 1999 to 2017 [3]. Using the queries presented in the surveys [3], [64], we updated both surveys from 2016 and 2017 until January 2023, respectively, and we identified the common works (see the 33 works in the middle part of the table). All of the works include human interaction, but only 36.36% of the works involve human assessment in the fitness function to evaluate candidate solutions. Of the works that include human assessment in the fitness function, only four of them involve industry in the evaluation. For example, Ghannem et al. [7] involve human assessment in the fitness function by combining structural similarity and designers' ratings to evaluate the model refactorings. Araujo et al. [10] present an architecture that combines interactive optimization and machine learning to address the next release problem in software engineering, incorporating human expertise and preferences for efficient requirement selection. Marculescu et al. [13] suggest optimizations for test cases using a dynamic fitness function that changes the weights of the objective values according to the assessment that is made by a human. The works by Kessentini et al. [16], [21] involve the human in the fitness function by interacting at the solution level by accepting/rejecting/modifying specific edit operations, and then computing the weighted probability of edit operations and target model elements.

Martinez et al. [9] completely replace the fitness function with a human, but they do not provide evidence supporting the idea that the human is beneficial. Their work makes a comparison between the human (combined with crossover and mutation operations) and Random Search. One of our previous works, Pérez et al. [17], completely replaces the fitness function with a human in a real-world industrial case study for Feature Location in Models (FLiM). Although the results show that the human as fitness function is beneficial, it is acknowledged that a hybrid approach could be explored to avoid human fatigue for more demanding problems. As Column 4 of Table II shows, none of the works explore a hybrid fitness function that combines simulations with humans, as this work does.

It is important to highlight that all of the works that combine human assessment with an algorithm in the fitness function [7], [10], [13], [14], [15], [16], [21], [22] do not address either BL or GSE, and they interleave the human effort with the assessment of the algorithm. For example, even though the works by Kessentini et al. [16], [21] study if the number of human interactions and time can be reduced by changing the algorithm, neither of them (see Column 5 of Table II) study whether the distribution of the human effort should be different in order to improve the quality of the solution (e.g., doing all human assessment first and then the algorithm assessment instead of interleaving the human-algorithm).

To address this lack, we propose three variants of the hybrid fitness function in order to study the influence in the quality of the solution of different ways of distributing the human effort during BL. The variants: 1) interleave the assessment of the candidate solutions between the human and the algorithm (distribution in the hybrid fitness function that is used by most of the related works); 2) rely on the assessment performed by the human, and then the assessment of the algorithm; or 3) the reverse (the assessment performed by the algorithm, and then the assessment of the human). Specifically, we study whether the selected variant influences the solution quality for SBMDE in the context of BL in GSE.

IV. Hybridizing the Fitness Function

Fig. 2 shows an overview of our approach, SimuHaFF (Simulation and Human as the Fitness Function), for BL in GSE. Our approach consists of an evolutionary algorithm where the fitness function is hybrid. It is comprised of the assessment of candidate solutions by both the score automatically provided by an algorithm using simulations and the score provided by a human. The upper part of Fig. 2 depicts the set of software models, which the algorithm takes as input to locate the bug. Afterwards, the population of scenarios is initialized by a random selection from the input models. The goal is to obtain a ranked list of simulation traces that are ordered by their relevance in locating the bug.

The hybrid fitness function assesses each of the scenarios. The fitness provided by the algorithm runs the simulations as described in Section II. In order to enable the human to assess candidate solutions, the approach displays the model fragments that are candidate solutions to be potential sources of bug. Then, the human provides the fitness score to each candidate solution, which is given on a scale of 1 to 7 with a maximum score of 7 (for the best candidates of having the bug) and a minimum of 1. The human evaluation is performed using a seven-point scale rather than a broad rating to reduce the fatigue of the developers, as recommended in [40].

In order to study how the distribution of the human effort in a hybrid fitness function influences the results, we propose three variants that take into account Takagi's [40] recommendations for avoiding human fatigue. These recommendations are the reduction of both the size of the population to be evaluated and the number of iterations (to 10 or 20). The variants (V1-V3) that we propose are:

Variant 1 interleaves simulations that are performed by the algorithm with human evaluations. The simulations are executed by pre-setting a time m (in seconds) each time that the human provides an evaluation. In total, the human performs n evaluations. Fig. 2-V_1 depicts how the simulations are interleaved with human evaluations in this variant. The upper part above the dotted line represents the human evaluation, whereas the lower part shows the evaluation that is automatically produced by the evolutionary algorithm using simulations.

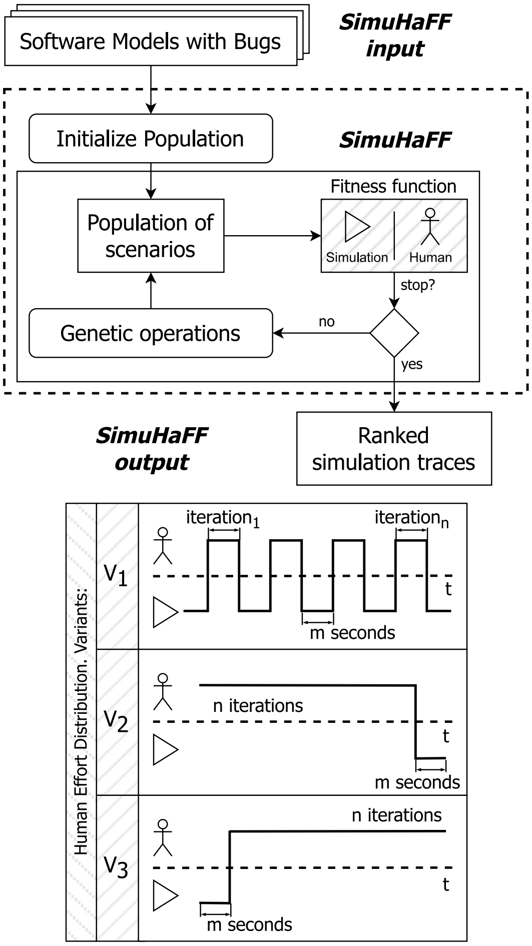


Fig. 2. Overview and variants for hybridizing the fitness function.

Variant 2 relies on n consecutive human evaluations and then launching the simulations of the algorithm for m seconds. Fig. 2-V_2 depicts the distribution of the human evaluations first and then the simulations of the algorithm.

Variant 3 reverses V2 by launching the simulations of the algorithm for m seconds and then the n consecutive human evaluations. Fig. 2-V_3 shows the distribution of this variant.

Once the candidate solutions are assessed by both the algorithm and the human, new scenarios are generated if the stop condition of the algorithm is not met. To do this, existing scenarios are selected using the wheel selection mechanism, where the selection of a scenario is inversely proportional to its relative fitness in the population. To modify the scenario, we use the widespread single-point crossover and random mutation (the most popular choice in SBMDE [65]).

Finally, when the stop condition of the algorithm is met, the output is a ranking of simulation traces, which are ordered by their relevance in locating the bug.

V. Evaluation

This section presents the evaluation of our work: the research questions that we aim to answer, the evaluation process

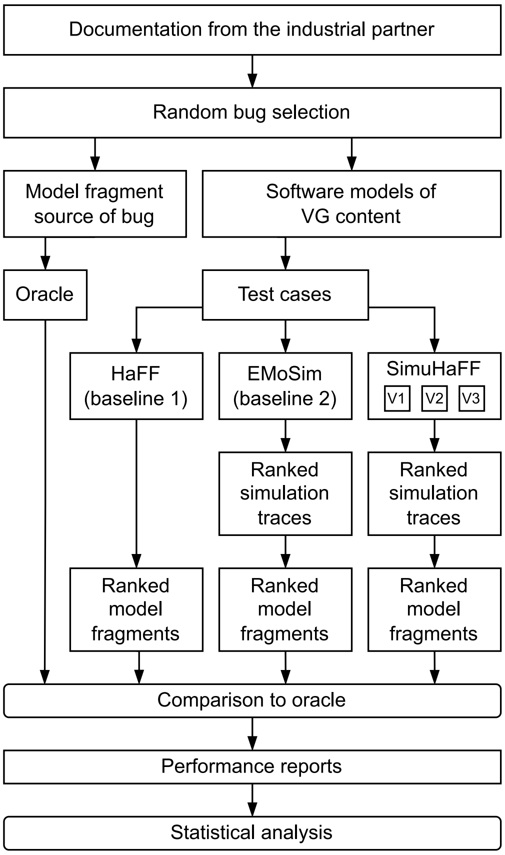


Fig. 3. Overview of the evaluation process.

(including recall, precision, and F-measure to measure the quality of the solutions; baselines; and statistical analysis), and the implementation details.

A. Research Questions

We aim to answer the following research questions:

RQ$_1$: *What is the performance in terms of solution quality using the three variants for distributing the human effort in the hybrid fitness function and the baselines in BL?*

RQ$_2$: *Is the difference in the quality of the solution between the variants and the baselines significant?*

RQ$_3$: *How much is the quality of the solution influenced by using each variant compared to each baseline?*

B. Planning and Execution

Fig. 3 shows an overview of the evaluation process to answer the research questions. The upper part of the figure shows the software models with bugs selected from the bug catalog provided by our industrial partner, which are the inputs for the test cases.

In the figure, the output of the test cases are two baselines and the three variants of our approach. The first baseline, HaFF, has the fitness function completely replaced by a human, and it outputs a sorted collection of model fragments that are considered to be the most relevant for the bug. The second baseline, EMoSim, has the unattended algorithm producing simulation traces as the fitness function, but it outputs a ranking of simulation traces as described in Section II. The trace contains all of the model elements that the interpreter has used at run-time during the simulation. All of the model elements that appear in the trace form the most relevant model fragment according to the trace for the bug. The three variants of our SimuHaFF approach work as described in Section IV to form the model fragments that are considered to be the most relevant for the bug.

Answering RQ$_1$: After executing the two baselines and the three variants of our approach, we take the first solution in the ranking for each of the bugs as suggested in [66]. Afterwards, we compare the solution against the oracle (i.e., the ground truth) in order to get a confusion matrix. The confusion matrix is a table that provides detailed information about the performance of a classification algorithm. In our work, each candidate solution is a model fragment that is composed of a subset of model elements (where the bug is to be found). The presence or absence of model elements is considered as a classification since the granularity is at the model element level. Accordingly, our confusion matrix distinguishes between two specific values: TRUE (present) or FALSE (absent).

A confusion matrix groups the results of a comparison into four separate categories: True positive (TP), when an element that is present in the predicted model fragment is also present in the model fragment from the oracle; True Negative (TN), when an element that is not present in the predicted model fragment is not present in the model fragment from the oracle; False Positive (FP), when an element that is present in the predicted model fragment is not present in the model fragment from the oracle; and False Negative (FN), when an element that is not present in the predicted model fragment is present in the model fragment from the oracle.

Once the TP+TN+FP+FN are calculated, some measurements are extracted from the confusion matrix in order to assess the performance in terms of the solution quality of the approach. For each baseline and variant of our approach, we specifically generate a report with three performance metrics (recall, precision, and F-measure) that are widely accepted in the software engineering research community [67]: Recall $\left(\frac{TP}{TP+FN}\right)$ measures the number of elements of the model fragment from the oracle that are correctly retrieved by the proposed model fragment; Precision $\left(\frac{TP}{TP+FP}\right)$ measures the number of elements from the proposed model fragment that are correct according to the ground truth (the oracle); and F-measure $\left(2 * \frac{Precision*Recall}{Precision+Recall}\right)$ corresponds to the harmonic mean of precision and recall.

Recall and precision values can range between 0 and 1. A recall value of 0 means that no single model element from the model fragment from the oracle is present in any of the model fragments of the predicted solution), whereas a value of 1 means that all of the model elements from the oracle are present in the predicted solution. A precision value of 0 means that no single model element from the model fragment predicted is present in the model fragment from the oracle, whereas a value of 1 means that all of the model elements from the predicted solution are present in the model fragment from the oracle. A value of 1 in precision and 1 in recall implies that both the predicted model fragment and the model fragment from the oracle are the same.

Answering RQ$_2$: To properly compare the approaches, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [44]. 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 more powerful than the others when working with real data [68]. The Quade test compares the results of multiple algorithms to determine whether there are significant differences among them. In addition, according to Conover [69], the Quade test has shown better results than the others when the number of algorithms is low (no more than four or five algorithms). However, with the Quade test, we cannot know which of the algorithms gives the best performance. In this case, the performance of each algorithm should be individually compared against all of the other alternatives. In order to do this, we perform an additional Holm's 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.

Answering RQ$_3$: 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 [44]. Thus, it is important to assess if an algorithm is statistically better than another and to assess the magnitude of the improvement. To assess how much the quality of the solution is influenced by using SimuHaFF compared to the baselines, the magnitude of the improvement should be assessed through *effect size* measures. For a non-parametric effect size measure, we used Cliff's delta [70]. Cliff's delta is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap with values ranging from -1 to 1. For example, when comparing Base2_EMoSim and V3_SimuHaFF a value of 0 means there is no difference, a value of -1 means that all samples in Base2_EMoSim are lower than all samples in V3_SimuHaFF, and a value of 1 means the opposite (all samples in Base2_EMoSim are higher than all samples in V3_SimuHaFF). Moreover, threshold values were defined in [45] for the interpretation of Cliff's delta effect size ($|d| < 0.147 \rightarrow$ "negligible"; $|d| < 0.33 \rightarrow$ "small"; $|d| < 0.474 \rightarrow$ "medium", $|d| \geq 0.474 \rightarrow$ "large"). We record a Cliff's delta value for each pair-wise comparison in recall, precision, and F-measure.

C. Implementation Details

In order to compare the baseline and variants of our approach fairly, we chose the parameters shown in Table III to calibrate the evolutionary algorithm and the fitness. As the table shows, the number of iterations the human performs is 10 and the number of candidate solutions (individuals) that the human evaluates per iteration is also 10. To determine the stop condition, we ran some prior tests to determine the convergence time. According to the tests, the time needed to converge was below 8 seconds for locating each bug. Therefore, we established the stop condition at 10 seconds (adding a margin to ensure convergence), ensuring that the approaches with the automatic fitness run long enough to obtain the best solutions. In this way, the number of possible simulations is defined by how many scenarios can be simulated during that time. Even though the population size is at most 100 scenarios, we only present to the human the best 10 in all of the variants and in the baseline that includes the human in the fitness function (HaFF) in order to prevent human fatigue [17]. We assume that the best candidate solutions are those with the lowest fitness value since they are ordered by ascending fitness. For the rest of parameters in the table, we used those settings that are commonly used in previous works [17], [43].

TABLE III
PARAMETER SETTINGS

	Parameter description	Value
Fitness		
HaFF and SimuHaFF	Iterations	10
	Individuals for subjective evaluation	10
EMoSim and SimuHaFF	$\%_{playerWin}$: Percentage goal of player's winnings	0.33
	$\%_{playerWinLife}$: Percentage goal of life left when player wins	0.35
	$Stop$: Time (in seconds) from convergence	10
Evolutionary algorithm		
HaFF	$Size$: Population size	10
EMoSim and SimuHaFF	$Size$: Population size	100
	$p_{hitByWeapon}$: Weapon hit probability	0.014
	$p_{hitByPlayer}$: Player hit probability	0.25
Crossover and mutation	μ: Number of parents	2
	λ: Number of offspring from μ parents	2
	$p_{crossover}$: Crossover probability	0.9
	$p_{mutation}$: Mutation probability	0.1

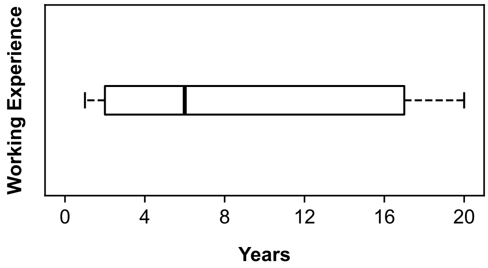


Fig. 4. Working experience of the participants.

In the evaluation, 29 professional video game developers participated. The recruitment of developers was carried out by our industrial partner, named Kraken Empire, who randomly selected the developers to participate. Kraken Empire is an independent game development studio, which is specialized in interactive 3D graphics applications, physics based simulations and real time systems. Fig. 4 shows the distribution of the developers' work experience. One out of every three has been developing video games for 15+ years, and the other two of them have developed video games for seven years or less. They all participated in the development of Kromaia, either from its inception (the most experienced developers) or creating new content for the game (19 developers). They were professionals holding a degree in computing and related areas. All of them were from Spain, except for one who was from Latin America.

With regard to the time that each developer spends on the evaluation, we chose a slot of two hours since it is the median duration of software engineering experiments [71]. Hence, each developer participates in the localization of four randomly assigned [72] bugs in total (one bug per variant of the approach, plus another one for pure baseline HaFF). Hence, 116 bugs (4*29) were located in total. The assignment was transparent to the developers, so they do not know whether they are locating the bug using one variant of the approach or the baseline HaFF. The decision to only select one bug to locate for each variant of the approach plus baseline was based on the time needed to rate an individual, which could last an average of 15 seconds. Thus, the time that each developer spends on the evaluation of individuals to locate the four bugs is 100 minutes: 10 (individuals for subjective evaluation) x 10 (iterations) = 100 (ratings/bug) * 15 (seconds/rating) = 1500 (seconds/bug) * 4 (bugs) = 6000 seconds (100 minutes).

The remaining 20 minutes are to provide a brief tutorial before the evaluation is started, to conduct a focus group when they finish and, indeed, to add a margin that ensures the completion of the evaluation. The brief tutorial mainly consisted of: 1) describing the objective of locating bugs in the context of Kromaia; 2) showing the main interface aspects to evaluate individuals; 3) providing a simple example (not using the bugs that will be located during the evaluation).

The experiment was conducted as scheduled. The session was held on December 14, 2022, at the San Jorge University facilities. None of the subjects in this experiment exceeded the established evaluation time limit. On average, the evaluation lasted 87 minutes, and the minimum and maximum duration were 63 and 100 minutes, respectively.

In order to collect the experimental data we spent about 1 hours and 17 minutes per participant on analyzing the results obtained. We analyzed a total of 29 participants, putting in a total of approximately 37.5 hours. The data related to participant characteristics (e.g., experience) were gathered automatically. The experiment was carried out according to the planned schedule. Therefore, there were no deviations during the execution of the experiment.

The evaluation of SimuHaFF and the baselines was done using identical gaming PCs with the same features. These were three ASUS ROG Strix laptops, using an Intel Core i7-6700HQ processor with 16 GiB of RAM, and running on a 64-bit Windows 10 host operating system. There are two reasons why the PCs were identical: (i) to avoid a potential bias when running the simulations, as technical specifications can influence the number of simulations run, and (ii) since the evaluation by professional video game developers is extremely time-consuming from a computer's perspective [40]. With these specifications, the automatic fitness part converges in 10 seconds and is capable of running more than 470k simulations on average.

The implementation uses the Java(TM) SE Runtime Environment (build 17.0.5), together with Kotlin as the programming language. For purposes of replicability, the implementation source code and the data (software models and oracles) as well as a screenshot of the graphical interface used by humans to evaluate the candidate solutions and the CSV files used as input in the statistical analysis are publicly available at the following URL: http://www.gamesoftwareengineering.com/tse23/bl-in-mgse.

TABLE IV
MEAN VALUES AND STANDARD DEVIATIONS FOR RECALL, PRECISION, AND F-MEASURE FOR EACH VARIANT.

	Recall	Precision	F-measure
V1_SimuHaFF	75.38 ± 3.98	65.64 ± 9.62	69.67 ± 5.17
V2_SimuHaFF	73.54 ± 4.58	75.86 ± 5.37	74.53 ± 3.66
V3_SimuHaFF	91.64 ± 4.29	84.84 ± 9.59	87.79 ± 5.74
Base1_HaFF	51.34 ± 7.38	58.38 ± 4.8	54.33 ± 5.03
Base2_EMoSim	67.33 ± 19.84	43.54 ± 19.66	49.86 ± 13.56

VI. RESULTS

In this section, we present the results obtained in the two baselines (Base1_HaFF and Base2_EMoSim) and in the three variants of our approach (V1-V3_SimuHaFF) in Kromaia.

A. Research Question 1

Table IV shows the mean values and standard deviations for recall, precision, and F-measure for each baseline and variant. All of the variants and the baseline that includes the human in the fitness function (Base1_HaFF) obtained better results than Base2_EMoSim. Specifically, V3_SimuHaFF (simulations generated by the algorithm first and then the human evaluations) yielded the best results, followed by V2_SimuHaFF and then V1_SimuHaFF.

RQ$_1$ answer. The results of all of the variants reveal that hybridizing the fitness functions pays off in the context of BL for GSE. The variants obtained an average value of 80.19 in recall and 75.45 in precision, with V3_SimuHaFF being the variant that obtained the best results (91.64% in recall and 84.84% in precision).

B. Research Question 2

The $p-Values$ obtained in the test are lower than $2.2x10^{-16}$ for recall, precision, and F-measure. Since the $p-Values$ are smaller than 0.05, we can state that there are differences among the algorithms for the performance indicators of recall, precision, and F-measure.

Table V shows the $p-Values$ of the Holm's post hoc analysis for each pair-wise comparison and performance indicator. All $p-Values$ obtained in precision and F-measure were smaller than their corresponding significance threshold value (0.05), indicating that the differences in performance between the three variants and the baselines are significant, except in recall when comparing *V1_SimuHaFF vs. Base2_EMoSim*, *V2_SimuHaFF vs. Base2_EMoSim* and *V1_SimuHaFF vs. V2_SimuHaFF*. Although these comparisons do not indicate significant differences in recall (meaning that the number of elements of the retrieved model fragment is similar), it is important to highlight that these comparisons indicate that there are significant differences in precision. This means that a higher number

TABLE V
HOLM'S POST HOC *P-VALUES* FOR EACH PAIR-WISE COMPARISON

	Recall	Precision	F-Measure
V1 vs Base1	$3x10^{-10}$	0.00051	$3x10^{-10}$
V1 vs Base2	0.063	$1.1x10^{-6}$	$2x10^{-8}$
V2 vs Base1	$3x10^{-10}$	$3x10^{-10}$	$3x10^{-10}$
V2 vs Base2	0.24	$4.8x10^{-8}$	$8.1x10^{-9}$
V3 vs Base1	$3x10^{-10}$	$3x10^{-10}$	$3x10^{-10}$
V3 vs Base2	$8.1x10^{-9}$	$1.5x10^{-9}$	$3x10^{-10}$
V1 vs V2	0.21	$2.4x10^{-6}$	$2.2x10^{-5}$
V1 vs V3	$3x10^{-10}$	$3x10^{-9}$	$3x10^{-10}$
V2 vs V3	$3x10^{-10}$	$5.4x10^{-5}$	$3x10^{-10}$

TABLE VI
EFFECT SIZE MEASURES FOR COMPARING EACH PAIR OF ALGORITHMS IN KROMAIA

	Cliff's Delta		
	Recall	Precision	F-Measure
V1 vs Base1	1 (large)	0.4482759 (medium)	0.9857313 (large)
V1 vs Base2	0.1557669 (small)	0.6575505 (large)	0.7526754 (large)
V2 vs Base1	1 (large)	1 (large)	1 (large)
V2 vs Base2	0.1034483 (negligible)	0.667063 (large)	0.8049941 (large)
V3 vs Base1	1 (large)	1 (large)	1 (large)
V3 vs Base2	0.7812128 (large)	0.8454221 (large)	0.9833532 (large)
V1 vs V2	0.2342449 (small)	-0.6147444 (large)	-0.529132 (large)
V1 vs V3	-1 (large)	-0.8287753 (large)	-0.9928656 (large)
V2 vs V3	-1 (large)	-0.5386445 (large)	-0.9548157 (large)

of model elements were correct according to the ground truth, which is relevant for the solution quality of the model fragment as shown in the corresponding F-measure value (which indicates significant differences).

RQ$_2$ answer. Since the Holm's post hoc $p-Values$ for F-measure that are shown in Table V are smaller than 0.05, we can state that there are significant differences between the variants and the baselines.

C. Research Question 3

Table VI shows the values of the effect size statistics of the pair-wise comparisons. Specifically, the table shows Cliff's Delta [70] values for recall, precision, and F-measure. From the results, we can determine how much the quality of the solution is influenced by using the variants of our approach compared to the baselines (HaFF and EMoSim) as well as the influence among the variants. The magnitude of improvement using any of the variants of our approach instead of the baselines can be interpreted as being large according to the magnitude scales [45] of the Cliff's Delta values, except for recall when comparing *V1_SimuHaFF vs. Base2_EMoSim*, *V2_SimuHaFF vs. Base2_EMoSim* and *V1_SimuHaFF vs. V2_SimuHaFF*.

RQ$_3$ answer. We can draw conclusions about how much the quality of the solution is influenced each variant of the hybrid fitness compared to each baseline from the results of Table VI. The results reveal that the magnitude of improvement in F-measure using any variant is large compared to the baselines according to the magnitude scales [45] of the Cliff's Delta values.

VII. DISCUSSION

This section presents: 1) the analysis of the results to understand why the quality of the solutions is influenced by the SimuHaFF variants; 2) the results of a focus group where we acquired feedback from the participants to determine whether the hybrid fitness function is accepted; and 3) the results of a survey where we obtained the opinion of video game developers on which SimuHaFF variant would obtain the best results (in terms of solution quality). Thus, we compare whether the SimuHaFF variant selected by the developers coincides with the SimuHaFF variant that obtains the best results.

We examined the results in order to understand why the human effort in the variants of SimuHaFF significantly influenced the quality of the solutions, especially using V3 (simulations of the algorithm first and then human evaluations). We detected that V3 provided the best candidate solutions that the algorithm locates as input to the human. Thus, the quality of the candidate solutions were even better (in all of the performance indicators) after the human evaluations than the solutions only produced by the algorithm.

In contrast, V2 (which reverses the V3 variant) did not obtain results as good as V3 because the human started the evaluations with candidate solutions that were randomly generated (in contrast to V3 where the human started with the best solutions obtained by the algorithm). Even though the quality of the initial candidate solutions improved after 10 iterations of human evaluations, it was not enough to obtain the best results due to the following: the genetic operations randomly produce new candidate solutions, which due to the randomness make solutions worse in some cases and the execution of the algorithm after the human evaluations produces solutions that are not the ones expected by the human (e.g., including incorrect model elements).

With regard to the results of V1 (interleaving simulations automatically produced by the algorithm with human evaluations), a similar effect occurred as in V2. Even though the algorithm provided the best solution possible and a single human evaluation influenced the selection that the algorithm made to produce new candidate solutions, the results that were displayed to the human in the next evaluation were not related to the ones that were previously scored with the highest fitness due to the algorithm. This ended up misleading the human and affecting their findings in every evaluation.

We ran a focus group to acquire feedback from the 29 professional video game developers of the industrial partner who participated as fitness function. The focus group was composed of the following open questions: (i) What do you think of the results of the approaches?; (ii) How do you feel about evaluating candidate solutions from simulation traces to locate bugs in video games?; (iii) How do you imagine the use of SimuHaFF in video games of other genres and in more complex video games?

The developers thought that the quality of the solution was better when they participated as fitness function than when they did not. They mentioned that doing the evaluations was intuitive and clear, and they enjoyed participating and using our approach even if it was to detect something that they would not have noticed without the approach. This is especially true in cases where the bugs comprise elements or properties of the models that are highly implicit or at a deep semantic level, since they are extremely complex to locate by a human alone.

The developers stated that the representation of the model fragments on the graphical interface was adequate. The heat map helped them to guide the bug localization. In addition, a potential improvement would be if the model were directly represented graphically in a model viewer that actually showed how it would look in-game, with the added highlighting of the parts that could be the source of a bug. Also, in their opinion, it would be important to always maintain a correlation with the textual model to be able to keep changes afterwards, or better yet, to be interactive to correct the model at run-time.

The developers also mentioned that SimuHaFF could be used for other video game genres and more complex problems if the video game is large (in terms of active assets) and the unit tests are complex or lack sufficient value or usefulness and therefore are not sufficiently representative. This is especially true for cases when a video game is an open game experience or the possibility of combinations between actions and results that the player performs are almost infinite. Some developers also stressed that regardless of whether or not you have the capacity (budget and effort required), it is better to have this type of assistance to find bugs than not to have it at all. In their opinion, even if the feedback from the approach is not entirely accurate, it can alert a developer to whether something may trigger unexpected behavior. The ideal for them and how they envision our approach in a real working environment is in the form of an integration with the model editor that simulates and tests models as they are built in an assisted manner. They mentioned that it would be worthwhile to have this approach even if the hit rate was low (e.g., it found 3 out of 50 bugs that they would not have to search for). This would be especially helpful for small studios or indie developers who have practically no time to test their games since they invest most of their resources in generating new content. Therefore, it would save them time and allow them to concentrate their efforts and focus on what is really important: creating their game and bringing it to market in a timely manner.

Furthermore, we emailed a survey to 20 video game developers to get their opinion on which SimuHaFF variant they thought would achieve the best results (in terms of solution quality). The survey included junior and senior video game developers (4 and 3, respectively) as well as academic experts (13), who are video game developers and also teach in the Design and development of video games degree program of San Jorge University. The survey featured a single-choice question, where the participants were asked to select the most promising variant among the SimuHaFF options. The question itself was accompanied by Fig. 2, which was supplemented with a descriptive caption explaining the content of the figure and each variant. Fourteen of the participants opted for interleaving simulations that are automatically produced by the evolutionary algorithm with human expert evaluations (V1), while four others preferred launching simulations that are generated by the algorithm first and then they themselves evaluate the results (V3). Only two thought that the best option for distributing human effort during bug localization would be to rely on several consecutive evaluations of the human expert before launching the simulations that are automatically obtained by the algorithm (V2). However, the results show that V3 performs best, followed by V2, V1, then B1_HaFF, and finally B2_EMoSim. Hence, the results are counter-intuitive for the video game developers who responded to the survey. Apparently to them, it made more sense for V1 to outperform V3. This is possibly influenced by how they envision a tool-assisted workflow.

VIII. Threats to Validity

To acknowledge the threats to the validity of our work, we use the classification suggested by De Oliveira et al. [73].

1) Conclusion Validity threats. We approached the *random variation* threat by considering 30 independent runs only in Baseline 2 (without human effort) for every single bug as suggested in [44]. However, that number of runs could not be contemplated given that the availability of humans is a limited resource, and the result of subsequent runs for the same bug would be influenced by the *learning effect*. For the *lack of a good descriptive analysis* threat, we applied the recall, precision, and F-measure performance metrics to analyze the confusion matrix obtained (although other metrics could be applied). We also applied statistical significance (the Quade test and Holm's post-hoc analysis) and effect size measurements (Cliff's Delta) following accepted guidelines [42]. We tackled the *lack of meaningful comparison baseline* issue by comparing the results obtained from our approach with two baselines: EMoSim and HaFF.

2) Internal Validity threats. To mitigate the threat of *poor parameter settings*, we used values from the SBSE literature. Default ones are adequate to measure performance of localization techniques, as indicated by Arcuri and Fraser [42]. We also used two main metrics (health and victory) to assess the fitness of a simulation as done in [46]. We handled the *lack of real problem instances* by selecting a commercial video game as the case study for the evaluation. Likewise, the problem artifacts were directly obtained from the video game developers and the documentation itself. Then, we randomly selected 29 bugs from the entire documentation. Afterwards, we also randomly assigned four different bugs to each developer in the evaluation. We did not participate in the selection of the developers to avoid *researcher bias* [74]. In addition, both the order of the variants of the approach and the baseline (which only uses the human as fitness function) and the set of bugs to be localized were randomly assigned to each engineer so that they would not know whether they were using a variant or baseline to locate a certain bug, hence mitigating the *imbalanced group of subjects* threat. We conducted the evaluation following a crossover design: the developers who participated in the oracle provided by the industrial partner were not involved in the evaluation, thus addressing the *learning effect* threat. We gave a briefing before starting the evaluation so that possible *understandability* issues could be minimized. The developers were not allowed to communicate with each other during the evaluation to avoid incurring in the *information exchange* threat. The number of iterations and individuals of the algorithm per bug was set to 10 as recommended in the literature [40] in order to reduce *fatigue impact*. Besides that, we established a total duration of the evaluation of two hours as a whole (including the briefing), in accordance with the average duration of experiments in software engineering [71].

3) Construct Validity threats. To prevent the *lack of assessing the validity of cost measures* threat, we made a fair comparison between the variants of our approach and the two baselines. Furthermore, we used three measurements (recall, precision, and F-measure) for the evaluation, which have been widely adopted by the software engineering research community [67].

4) External Validity threats. We evaluated our approach in a commercial video game, whose instances are collected from real-world problems, to avoid the *lack of a clear object selection strategy* threat. To mitigate the *generalization* threat, we designed our approach to be generic and applicable not only to our industrial case study but also for locating bugs in other different video games. To apply our approach in other video games, three main ingredients are required as in other SBSE approaches: encoding, operations, and fitness function. The operations are widespread crossover and mutation. The encoding and the fitness function depend on the simulated player. We can apply our approach to other video games where simulated players are available. These simulated players can be found in well-known gaming genres that include racing, first-person shooter, and real-time strategy. For those cases where there is no simulated player, the developers should ponder the tradeoff of the cost of developing the simulated player and the benefits of locating bugs using our approach. Nevertheless, our approach should be replicated with other video games before assuring its generalization.

IX. Conclusion

Recently, human participation in Search-Based Software Engineering (SBSE) has proven to be useful for obtaining more favorable solutions. However, in order to efficiently explore large complex problem spaces, humans cannot handle the same number of solutions as an algorithm since humans are not immune to fatigue.

In this work, we have proposed a hybrid fitness function that is the first to combine human effort with human simulations for more demanding problems in the context of BL in software models for the video games of GSE. Moreover, we have proposed three variants of the hybrid fitness function that distribute the human effort in different ways in order to study their influence on the quality of the solutions. The evaluation was performed on a commercial video game (Kromaia) where 29 professional video game developers were involved, acting as the human component of the fitness function. As baselines, we replaced the hybrid fitness function with a fitness function that only uses a human and a fitness function that is automatically calculated.

Our proposed hybrid fitness function outperformed the results of the best baseline by 33.46% in F-measure. Furthermore, a focus group confirmed the professional video game developers' acceptance of the hybrid fitness function since it helped them to reduce the amount of manual work and to minimize the number of bugs that go unnoticed. The variant that obtained the best results was not only counter-intuitive with an initial survey that we did with video game developers, but it was also counter-intuitive with previous works.

It is important to highlight that our work has implications from both an academic perspective in software engineering and from a practical standpoint for professional video game developers. From the academic perspective in software engineering, our work shows that the distribution of the human effort (interleaving the algorithm with human evaluations), which is both preferred by video game developers and selected by default in most iSBSE works, does not obtain the best results. This can inspire other academics to explore other distributions of the human effort (such as the distributions of the human effort that are studied in this work) to improve their results. From the practical standpoint of professional video game developers, there is a lack of technical solutions that locate bugs and real-world experience. This is hard to obtain since the majority of related works use academic data and other video game studios do not share the details of their technical solutions to develop and maintain their commercial products. Works like ours may be the path to compensate for this lack, and may motivate other video game developers to reduce the amount of tedious manual work during the location of bugs.

Our results not only help video game developers to locate bugs, but it can also inspire other SBSE researchers to bring hybrid fitness functions to other software engineering tasks instead of using a default approach for distributing the human effort. In fact, part of our future work is to explore hybrid fitness functions in other software engineering tasks and domains.

References

[1] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, 2012, doi: 10.1145/2379776.2379787.

[2] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Achieving feature location in families of models through the use of search-based software engineering," *IEEE Trans. Evol. Comput.*, vol. 22, no. 3, pp. 363–377, Jun. 2018.

[3] A. Ramírez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Trans. Softw. Eng.*, vol. 45, no. 8, pp. 760–781, Aug. 2019.

[4] Z. Wang, X.-Y. Fan, Y.-G. Zou, and X. Chen, "Genetic algorithm based multiple faults localization technique," *J. Softw.*, vol. 27, no. 4, pp. 879–900, 2016.

[5] B. Marculescu, R. Feldt, R. Torkar, and S. M. Poulding, "An initial industrial evaluation of interactive search-based testing for embedded software," *Appl. Soft Comput.*, vol. 29, pp. 26–39, Apr. 2015, doi: 10.1016/j.asoc.2014.12.025.

[6] R. Feldt, "An interactive software development workbench based on biomimetic algorithms," Dept. of Comput. Eng., Chalmers Univ. of Technol., Gothenburg, Tech. Rep. 02–16, 2002. [Online]. Available: http://www.cse.chalmers.se/feldt/publications/feldt_2002_wise_tech_report.html

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

[8] B. Amal, M. Kessentini, S. Bechikh, J. Dea, and L. B. Said, "On the use of machine learning and search-based software engineering for ill-defined fitness function: A case study on software refactoring," in

Search-Based Software Engineering, C. Le Goues and S. Yoo, Eds., Cham, Switzerland: Springer International Publishing, 2014, pp. 31–45.

[9] J. Martinez et al., *Variability Management and Assessment for User Interface Design*. Cham, Switzerland: Springer International Publishing, 2017, pp. 81–106.

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

[11] W. Kessentini, M. Wimmer, and H. Sahraoui, "Integrating the designer in-the-loop for metamodel/model co-evolution via interactive computational search," in *Proc. 21th ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, New York, NY, USA: ACM, 2018, pp. 101–111.

[12] J. Martinez et al., "Towards estimating and predicting user perception on software product variants," in *New Opportunities for Software Reuse*, R. Capilla, B. Gallina, and C. Cetina, Eds., Cham, Switzerland: Springer International Publishing, 2018, pp. 23–40.

[13] B. Marculescu, R. Feldt, R. Torkar, and S. Poulding, "Transferring interactive search-based software testing to industry," *J. Syst. Softw.*, vol. 142, pp. 156–170, Aug. 2018.

[14] C. V. Bindewald, W. M. Freire, A. M. M. M. Amaral, and T. E. Colanzi, "Towards the support of user preferences in search-based product line architecture design: An exploratory study," in *Proc. 33rd Brazilian Symp. Softw. Eng. (SBES)*, New York, NY, USA: ACM, 2019, pp. 387–396.

[15] V. Alizadeh, M. Kessentini, M. W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "An interactive and dynamic search-based approach to software refactoring recommendations," *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 932–961, Sep. 2020, doi: 10.1109/TSE.2018.2872711.

[16] W. Kessentini and V. Alizadeh, "Interactive metamodel/model co-evolution using unsupervised learning and multi-objective search," in *Proc. ACM/IEEE 23rd Int. Conf. Model Driven Eng. Lang. Syst. (MoDELS)*, E. Syriani, H. A. Sahraoui, J. de Lara, and S. Abrahão, Eds., New York, NY, USA: ACM, 2020, pp. 68–78, doi: 10.1145/3365438.3410966.

[17] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Empowering the human as the fitness function in search-based model-driven engineering," *IEEE Trans. Softw. Eng.*, vol. 48, no. 11, pp. 4553–4568, Nov. 2022.

[18] A. Ramírez, P. Delgado-Pérez, K. J. Valle-Gómez, I. Medina-Bulo, and J. R. Romero, "Interactivity in the generation of test cases with evolutionary computation," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Kraków, Poland. Piscataway, NJ, USA: IEEE Press, 2021, pp. 2395–2402, doi: 10.1109/CEC45853.2021.9504786.

[19] F. H. Kuviatkovski, W. M. Freire, A. M. M. M. Amaral, T. E. Colanzi, and V. D. Feltrim, "Evaluating machine learning algorithms in representing decision makers in search-based PLA," in *Proc. IEEE 19th Int. Conf. Softw. Archit. Companion (ICSA)*, Honolulu, HI, USA. Piscataway, NJ, USA: IEEE Press, 2022, pp. 68–75, doi: 10.1109/ICSA-C54293.2022.00057.

[20] P. Delgado-Pérez, A. Ramírez, K. J. Valle-Gómez, I. Medina-Bulo, and J. R. Romero, "InterEvo-Tr: Interactive evolutionary test generation with readability assessment," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 2580–2596, Apr. 2023.

[21] W. Kessentini and V. Alizadeh, "Semi-automated metamodel/model co-evolution: A multi-level interactive approach," *Softw. Syst. Model.*, vol. 21, no. 5, pp. 1853–1876, 2022, doi: 10.1007/s10270-022-00978-2.

[22] W. M. Freire, C. T. Rosa, A. M. M. M. Amaral, and T. E. Colanzi, "Validating an interactive ranking operator for NSGA-II to support the optimization of software engineering problems," in *Proc. 36th Brazilian Symp. Softw. Eng. (SBES)*, M. de Almeida Maia, F. A. Dorça, R. D. Araujo, C. von Flach, E. Y. Nakagawa, and E. D. Canedo, Eds., New York, NY, USA: ACM, 2022, pp. 337–346, doi: 10.1145/3555228.3555232.

[23] T. Yue, S. Ali, H. Lu, and K. Nie, "Search-based decision ordering to facilitate product line engineering of cyber-physical system," in *Proc. 4th Int. Conf. Model-Driven Eng. Softw. Develop. (MODELSWARD)*, 2016, pp. 691–703.

[24] H. Lu, T. Yue, S. Ali, and L. Zhang, "Nonconformity resolving recommendations for product line configuration," in *Proc. IEEE Int. Conf. Softw. Testing, Verification Validation (ICST)*, 2016, pp. 57–68.

[25] H. L. Jakubovski Filho, T. N. Ferreira, and S. R. Vergilio, "Preference based multi-objective algorithms applied to the variability testing of software product lines," *J. Syst. Softw.*, vol. 151, pp. 194–209, May 2019.

[26] D. N. A. d. Silva, "Adaptation oriented test data generation for adaptive systems," in *Proc. 15th Iberian Conf. Inf. Syst. Technologies* (CISTI), 2020, pp. 1–7.

[27] Y. Lin, X. Peng, Y. Cai, D. Dig, D. Zheng, and W. Zhao, "Interactive and guided architectural refactoring with search-based recommendation," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, New York, NY, USA: ACM, 2016, pp. 535–546.

[28] L. Van Rooijen and H. Hamann, "Requirements specification-by-example using a multi-objective evolutionary algorithm," in *Proc. IEEE 24th Int. Requirements Eng. Conf. Workshops (REW)*, 2016, pp. 3–9.

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

[30] E. Batot and H. Sahraoui, "A generic framework for model-set selection for the unification of testing and learning mde tasks," in *Proc. ACM/IEEE 19th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, New York, NY, USA: ACM, 2016, pp. 374–384.

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

[32] P. Gómez-Abajo, E. Guerra, and J. de Lara, "A domain-specific language for model mutation and its application to the automated generation of exercises," *Comput. Lang., Syst. Struct.*, vol. 49, pp. 152–173, Sep. 2017.

[33] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, and N. Paoletti, "Designing robust software systems through parametric Markov chain synthesis," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, 2017, pp. 131–140.

[34] A. Kolchin, "Interactive method for cumulative analysis of software formal models behavior," *Problems Program.*, vol. 2–3, nos. 23, pp. 115–123, 2018.

[35] S. Procter and L. Wrage, "Guided architecture trade space exploration: Fusing model based engineering design by shopping," in *Proc. ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, 2019, pp. 117–127.

[36] T. L. Calvar, F. Chhel, F. Jouault, and F. Saubion, "Toward a declarative language to generate explorable sets of models," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput. (SAC)*, Limassol, Cyprus, C. Hung and G. A. Papadopoulos, Eds., New York, NY, USA: ACM, 2019, pp. 1837–1844, doi: 10.1145/3297280.3297461.

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

[38] B. Alkhazi, C. Abid, M. Kessentini, D. Leroy, and M. Wimmer, "Multi-criteria test cases selection for model transformations," *Automated Softw. Eng.*, vol. 27, nos. 1–2, pp. 91–118, Jun. 2020.

[39] B. Alkhazi, C. Abid, M. Kessentini, and M. Wimmer, "On the value of quality attributes for refactoring ATL model transformations: A multi-objective approach," *Inf. Softw. Technol.*, vol. 120, Apr. 2020, Art. no. 106243.

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

[41] A. Ampatzoglou and I. Stamelos, "Software engineering research for computer games: A systematic review," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 888–901, 2010.

[42] A. Arcuri and G. Fraser, "Parameter tuning or default values? An empirical investigation in search-based software engineering," *Empirical Softw. Eng.*, vol. 18, no. 3, pp. 594–623, 2013.

[43] R. Casamayor, L. Arcega, F. Pérez, and C. Cetina, "Bug localization in game software engineering: Evolving simulations to locate bugs in software models of video games," in *Proc. 25th Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*, Montreal, QC, Canada, E. Syriani, H. A. Sahraoui, N. Bencomo, and M. Wimmer, Eds., New York, NY, USA: ACM, 2022, pp. 356–366, doi: 10.1145/3550355.3552440.

[44] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Testing, Verification Rel.*, vol. 24, no. 3, pp. 219–250, May 2014.

[45] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys," in *Proc. Annu. Meeting Florida Assoc. Institutional Res.*, 2006, pp. 1–33.

[46] D. Blasco, J. Font, M. Zamorano, and C. Cetina, "An evolutionary approach for generating software models: The case of Kromaia in game software engineering," *J. Syst. Softw.*, vol. 171, Jan. 2021, Art. no. 110804, doi: 10.1016/j.jss.2020.110804.

[47] L. Burgueño, J. Troya, M. Wimmer, and A. Vallecillo, "Static fault localization in model transformations," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 490–506, May 2015.

[48] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, “An automated model based testing approach for platform games,” in *Proc. 18th ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. (MoDELS)*, Ottawa, ON, Canada, T. Lethbridge, J. Cabot, and A. Egyed, Eds. New York, NY, USA: IEEE Computer Society, 2015, pp. 426–435, doi: 10.1109/MODELS.2015.7338274.

[49] J. Sánchez Cuadrado, E. Guerra, and J. Lara, “Static analysis of model transformations,” *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 868–897, Sep. 2017.

[50] J. Sánchez Cuadrado, E. Guerra, and J. Lara, “Quick fixing ATL transformations with speculative analysis,” *Softw. Syst. Model.*, vol. 17, no. 3, pp. 779–813, Jul. 2018, doi: 10.1007/s10270-016-0541-1.

[51] J. Troya, S. Segura, J. A. Parejo, and A. Ruiz-Cortés, “Spectrum-based fault localization in model transformations,” *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 1–50, Sep. 2018, doi: 10.1145/3241744.

[52] S. Ariyurek, A. Betin-Can, and E. Surer, “Automated video game testing using synthetic and humanlike agents,” *IEEE Trans. Games*, vol. 13, no. 1, pp. 50–67, Mar. 2021.

[53] Y. Zheng et al., “Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning,” in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA. Piscataway, NJ, USA: IEEE Press, 2019, pp. 772–784, doi: 10.1109/ASE.2019.00077.

[54] S. Ariyurek, A. Betin-Can, and E. Sürer, “Enhancing the Monte Carlo tree search algorithm for video game testing,” in *Proc. IEEE Conf. Games (CoG)*, Osaka, Japan. Piscataway, NJ, USA: IEEE Press, 2020, pp. 25–32, doi: 10.1109/CoG47356.2020.9231670.

[55] X. Cheng, N. Liu, L. Guo, Z. Xu, and T. Zhang, “Blocking bug prediction based on XGBoost with enhanced features,” in *Proc. 44th IEEE Annu. Comput., Softw., Appl. Conf. (COMPSAC)*, Madrid, Spain. Piscataway, NJ, USA: IEEE Press, 2020, pp. 902–911, doi: 10.1109/COMPSAC48688.2020.0-152.

[56] J. Zhang, R. Xie, W. Ye, Y. Zhang, and S. Zhang, “Exploiting code knowledge graph for bug localization via bi-directional attention,” in *Proc. 28th Int. Conf. Program Comprehension,* Seoul, Republic of Korea. New York, NY, USA: ACM, 2020, pp. 219–229, doi: 10.1145/3387904.3389281.

[57] L. Arcega, J. Font, Ø. Haugen, and C. Cetina, “Bug localization in model-based systems in the wild,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 10:1–10:32, 2022, doi: 10.1145/3472616.

[58] R. Ferdous, F. M. Kifetew, D. Prandi, I. S. W. B. Prasetya, S. Shirzadehhajimahmood, and A. Susi, “Search-based automated play testing of computer games: A model-based approach,” in *Proc. 13th Int. Symp., Search-Based Softw. Eng. (SSBSE)*, Bari, Italy, U. O’Reilly and X. Devroey, Eds., vol. 12914. Cham, Switzerland: Springer-Verlag, 2021, pp. 56–71, doi: 10.1007/978-3-030-88106-1_5.

[59] S. Quach, M. Lamothe, B. Adams, Y. Kamei, and W. Shang, “Evaluating the impact of falsely detected performance bug-inducing changes in JIT models,” *Empirical Softw. Eng.*, vol. 26, no. 5, 2021, Art. no. 97, doi: 10.1007/s10664-021-10004-6.

[60] A. Ciborowska and K. Damevski, “Fast changeset-based bug localization with BERT,” in *Proc. 44th IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, Pittsburgh, PA, USA. New York, NY, USA: ACM, 2022, pp. 946–957, doi: 10.1145/3510003.3510042.

[61] A. Khanfir, “Effective and scalable fault injection using bug reports and generative language models,” in *Proc. 30th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*, Singapore, A. Roychoudhury, C. Cadar, and M. Kim, Eds., New York, NY, USA: ACM, 2022, pp. 1790–1794, doi: 10.1145/3540250.3558907.

[62] H. Liang, D. Hang, and X. Li, “Modeling function-level interactions for file-level bug localization,” *Empirical Softw. Eng.*, vol. 27, no. 7, 2022, Art. no. 186, doi: 10.1007/s10664-022-10237-z.

[63] R. Tufano, S. Scalabrino, L. Pascarella, E. Aghajani, R. Oliveto, and G. Bavota, “Using reinforcement learning for load testing of video games,” in *Proc. 44th Int. Conf. Softw. Eng. (ICSE)*, Pittsburgh, PA, USA, 2022, pp. 2303–2314.

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

[65] F. Pérez, T. Ziadi, and C. Cetina, “Utilizing automatic query reformulations as genetic operations to improve feature location in software models,” *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 713–731, Feb. 2022, doi: 10.1109/TSE.2020.3000520.

[66] H. Ishibuchi, Y. Nojima, and Tsutomu Doi, “Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures,” in *Proc. IEEE Int. Conf. Evol. Comput.*, 2006, pp. 1143–1150.

[67] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, 1986.

[68] S. García, A. Fernández, J. Luengo, and F. Herrera, “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power,” *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.

[69] W. Conover, *Practical Nonparametric Statistics* (Wiley Series in Probability and Statistics), 3rd ed. New York, NY, USA: Wiley, 1999.

[70] N. Cliff, *Ordinal Methods for Behavioral Data Analysis.* New York, NY, USA: Lawrence Erlbaum Associates, 1996.

[71] D. I. K. Sjoeberg et al., “A survey of controlled experiments in software engineering,” *IEEE Trans. Softw. Eng.*, vol. 31, no. 9, pp. 733–753, Sep. 2005.

[72] H. J. Seltman, *Experimental Design and Analysis.* Pittsburgh, PA, USA: Carnegie Mellon Univ., 2012. [Online]. Available: http://www.stat.cmu.edu/hseltman/309/Book/Book.pdf

[73] M. de Oliveira Barros and A. C. D. Neto, “Threats to validity in search-based software engineering empirical studies,” *RelaTe-DIA*, vol. 5, Jan. 2011.

[74] R. Feldt and A. Magazinius, “Validity threats in empirical software engineering research – An initial survey,” in *Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng. (SEKE)*, 2010, pp. 374–379.

Rodrigo Casamayor is working toward the Ph.D. degree in computer science with the Polytechnic University of Valencia. He is a Researcher with the SVIT Research Group (https://svit.usj.es) at San Jorge University. His current research interests include bug localization and game software engineering. He publishes his research results and participates in high-level international software engineering conferences, such as the International Conference on Model Driven Engineering Languages and Systems (MODELS).

Carlos Cetina received the Ph.D. degree in computer science from the Polytechnic University of Valencia. He is an Associate Professor with San Jorge University and the Head of the SVIT Research Group. His research focuses on software product lines and model-driven development. His research results have reshaped software development in world-leading industries from heterogeneous domains ranging from induction hob firmware to train control and management systems. For more information, see http://carloscetina.com.

Óscar Pastor (Member, IEEE) is currently a Full Professor and the Director of the PROS Research Center at the Polytechnic University of Valencia. With a strong background in conceptual modeling, model-driven development and their practical applications in information systems design and development, he is currently leading a multidisciplinary project linking information systems and bioinformatics with designing and implementing tools for conceptual modeling-based interpretation of human genome information.

Francisca Pérez received the Ph.D. degree in computer science from the Polytechnic University of Valencia. She is an Associate Professor with the SVIT Research Group (https://svit.usj.es) at San Jorge University. Her research interests include model-driven development, collaborative information retrieval, search-based software engineering, and variability modeling. She publishes her research results and participates in high-level international software engineering conferences and journals, such as IEEE TRANSACTIONS ON SOFTWARE ENGINEERING (TSE), the *Automated Software Engineering (AUSE)* journal, the *Information and Software Technology (IST)* journal, and the *Journal of Systems and Software (JSS)*. For more information, see http://franciscaperez.com.