



March 2026

Directors:

Dra. Lorena **Arcega Rodríguez**
Dr. Óscar **Pastor López**



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Bug Localization in Software Models in Video Games

March 2026

Author: Isis Roca Mainer

Directors: Dra. Lorena Arcega Rodriguez
Dr. Óscar Pastor López

Acknowledgments

First of all, I would like to express my deepest gratitude to my directors, Dra. Lorena Arcega and Dr. Óscar Pastor, for their invaluable guidance, support, and encouragement throughout this journey.

I also extend my appreciation to my colleagues from the SVIT research group and EARTE. The collaborative spirit, motivation, and enthusiasm of the team have made this experience not only productive but also fun. Every achievement, no matter how small, has been a reason to celebrate, and I'm grateful to be part of such an inspiring team.

A special thanks to Chema for his unwavering support and understanding and for making my life easier during these years. His patience and encouragement have been fundamental in helping me navigate this journey.

To my family, my parents instilled in me the value of education, continuous learning, and self-improvement. Their example has been my foundation. To my sister, Surya, my role model, whose unwavering support and encouragement have meant the world to me.

Finally, to my friends, thank you for understanding that my absence over these years never meant that I was not there for you. Your patience and support have been invaluable.

Abstract

The video game industry has evolved into a highly profitable sector with increasingly complex development processes, leading to the emergence of Game Software Engineering. In addition, the adoption of Model-Driven Development (MDD) in game development is growing, however, bug localization methods remain primarily focused on source code, overlooking the critical role of software models. Bugs in video games often arise from intricate interactions between game mechanics, player behavior, and the game environment, necessitating specialized bug localization approaches. This thesis aims to develop bug localization techniques tailored to software models in game development.

This thesis compendium comprises three studies, ranging from an investigation into the current state of bug localization in video games to the proposal and evaluation of a novel approach that advances the state of the art. The first study explores the impact of Search-Based Software engineering on MDD maintenance tasks, analyzing how different search parameter configurations influence model fragment location performance. The second study compares MDD and Code-centric Development (CcD) in terms of performance, productivity, and satisfaction, finding that while MDD increases user satisfaction, professionals perform better in CcD, revealing a knowledge gap in MDD bug localization. The third study proposes an approach for bug localization in video games using MDD, demonstrating that taking into account model interactions improves bug localization compared to baseline methods.

Beyond entertainment, video games are increasingly influencing other domains such as film and television production, architectural visualization, simulation-

based training, and even the infrastructure of future virtual worlds envisioned by initiatives like those of the European Union. This growing relevance further underscores the need for robust software engineering practices tailored to the unique challenges of game development.

Resumen

La industria del videojuego ha evolucionado hasta convertirse en un sector muy rentable con procesos de desarrollo cada vez más complejos, lo que ha propiciado la aparición de la Ingeniería de Software de Juegos. Además, la adopción del Desarrollo Orientado a Modelos (MDD) en el desarrollo de videojuegos es cada vez mayor; sin embargo, los métodos de localización de errores siguen centrándose principalmente en el código fuente, pasando por alto el papel fundamental de los modelos de software. Los errores en los videojuegos surgen a menudo de intrincadas interacciones entre la mecánica del juego, el comportamiento del jugador y el entorno del juego, lo que requiere enfoques especializados de localización de errores. Esta tesis pretende desarrollar técnicas de localización de errores adaptadas a los modelos de software en el desarrollo de videojuegos.

Este compendio de tesis comprende tres estudios, que van desde una investigación sobre el estado actual de la localización de bugs en videojuegos hasta la propuesta y evaluación de un enfoque novedoso que avanza el estado del arte. El primer estudio explora el impacto de la ingeniería de software basada en la búsqueda en las tareas de mantenimiento de MDD, analizando cómo influyen las distintas configuraciones de los parámetros de búsqueda en el rendimiento de la localización de fragmentos del modelo. El segundo estudio compara el MDD y el desarrollo centrado en el código (Code-centric Development, CcD) en términos de rendimiento, productividad y satisfacción, descubriendo que, mientras que el MDD aumenta la satisfacción del usuario, los profesionales obtienen mejores resultados en el CcD, lo que revela una laguna de conocimiento en la localización de fallos en el MDD. El tercer estudio propone un enfoque

para la localización de errores en videojuegos mediante MDD, demostrando que tener en cuenta las interacciones de los modelos mejora la localización de errores en comparación con los métodos de referencia.

Más allá del entretenimiento, los videojuegos influyen cada vez más en otros ámbitos, como la producción de cine y televisión, la visualización arquitectónica, la formación basada en la simulación e incluso la infraestructura de los futuros mundos virtuales previstos por iniciativas como las de la Unión Europea. Esta creciente relevancia subraya aún más la necesidad de contar con sólidas prácticas de ingeniería de software adaptadas a los retos únicos del desarrollo de juegos.

Resum

La indústria del videojoc ha evolucionat fins a convertir-se en un sector molt rendible amb processos de desenrotllament cada vegada més complexos, la qual cosa ha propiciat l'aparició de l'Enginyeria de Programari de Jocs. A més, l'adopció del Desenrotllament Orientat a Models (MDD) en el desenrotllament de videojocs és cada vegada major; no obstant això, els mètodes de localització d'errors continuen centrant-se principalment en el codi font, passant per alt el paper fonamental dels models de programari. Els errors en els videojocs sorgixen sovint d'intricades interaccions entre la mecànica del joc, el comportament del jugador i l'entorn del joc, la qual cosa requereix enfocaments especialitzats de localització d'errors. Esta tesi pretén desenrotllar tècniques de localització d'errors adaptades als models de programari en el desenrotllament de videojocs.

Este compendi de tesi comprén tres estudis, que van des d'una investigació sobre l'estat actual de la localització de bugs en videojocs fins a la proposta i avaluació d'un enfocament nou que avança l'estat de l'art. El primer estudi explora l'impacte de l'enginyeria de programari basada en la busca en les tasques de manteniment de MDD, analitzant com influïxen les diferents configuracions dels paràmetres de busca en el rendiment de la localització de fragments del model. El segon estudi compara el MDD i el desenrotllament centrat en el codi (Code-centric Development, CcD) en termes de rendiment, productivitat i satisfacció, descobrint que, mentres que el MDD augmenta la satisfacció de l'usuari, els professionals obtenen millors resultats en el CcD, la qual cosa revela una llacuna de coneixement en la localització de fallades en el MDD. El tercer estudi proposa un enfocament per a la localització d'errors en

videojocs mitjançant MDD, demostrant que tindre en compte les interaccions dels models millora la localització d'errors en comparació amb els mètodes de referència.

Més enllà de l'entreteniment, els videojocs influïxen cada vegada més en altres àmbits, com la producció de cinema i televisió, la visualització arquitectònica, la formació basada en la simulació i fins i tot la infraestructura dels futurs mons virtuals previstos per iniciatives com les de la Unió Europea. Esta creixent rellevància subratlla encara més la necessitat de comptar amb sòlides pràctiques d'enginyeria de programari adaptades als reptes únics del desenrotllament de jocs.

Contents

Acknowledgments	iii
Abstract - Resumen - Resum	v
Contents	xi
I Introduction	1
II Compendium of Articles	21
1 On the Interaction between the Search Parameters and the Nature of the Search Problems in Search-Based Model-Driven Engineering	23
2 Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games	63
3 Co-evolving Scenarios and Simulated Players to Locate Bugs that arise from the Interaction of Software Models of Video Games	85

III	Discussion	121
IV	Conclusions	129

Part I

Introduction

Introduction

This chapter presents the context, motivation, and objectives of this thesis. It also provides an overview of the research process and details the applied methodology. Finally, the chapter concludes with a description of the thesis structure and a summary of the scientific articles included in the compendium.

Context

The video game industry has emerged as a significant and highly profitable segment of the software sector¹. As video games continue to gain popularity, their development has become increasingly complex, requiring sophisticated software to manage diverse aspects such as game mechanics, graphics, physics, and player interactions. This complexity has led to a specialized field known as Game Software Engineering (GSE), which focuses on creating and maintaining software specifically designed for video games [1]. In this context, video game development is increasingly adopting Model-Driven Development (MDD), which facilitates collaboration among heterogeneous profiles, such as programmers, artists, and designers, throughout the development process.

Game Software Engineering

GSE involves applying Software Engineering techniques and methodologies within the context of video game development. Commercial video games are complex products that differ significantly from traditional software due to their unique requirements and components. Game development is a multi-disciplinary field that allows professionals to contribute in various specialized areas. The main branches include:

- In **Game Design**, specialists focus on different aspects of the game's conceptualization and structure. *Concept Design* involves developing the initial idea and theme, setting the foundation for the game's world. *Level Design* is responsible for creating environments, defining gameplay flow, and ensuring an engaging player experience. *Game Mechanics* specialists define the rules, systems, and interactions that shape how players engage with the game.
- In **Art and Animation**, professionals handle the visual aspects of the game. *2D Art* includes creating sprites, backgrounds, and UI elements, while *3D Modeling* focuses on building characters, environments, and objects in three dimensions. *Animation* brings these assets to life through movement and realistic motion. *Texture and Lighting* specialists enhance the game's visuals by applying detailed textures and optimizing lighting effects.

¹T. B. R. Company, Gaming global market report 2023, <https://www.thebusinessresearchcompany.com/report/gaming-global-market-report>.

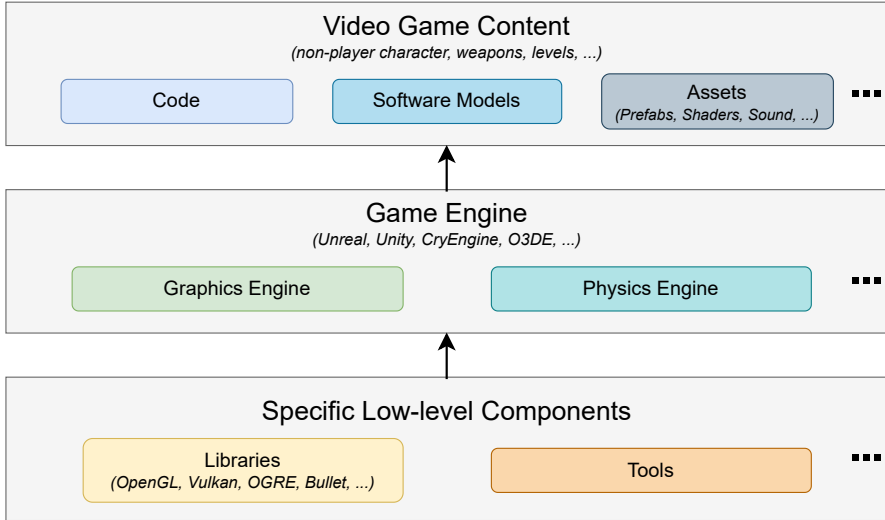


Figure 1: Overview of video game software artifacts.

- **Audio**, is a vital element in creating an immersive experience. *Sound Design* ensures that every action in the game is accompanied by appropriate sound effects. *Music Composition* creates background scores and themes that enhance the game’s atmosphere. *Voice Acting* provides character dialogue, adding depth to storytelling and player engagement.
- **Software** is essential for implementing the game’s functionality and mechanics which integrates and coordinates all production elements in the final playable application. *Gameplay Software* focuses on coding mechanics and interactive features. *Graphics Software* works on rendering, shaders, and visual effects to optimize visual fidelity. *AI Software* develops intelligent behaviors for non-player characters (NPCs), making them more responsive and engaging. *Networking* specialists build and maintain online multiplayer functionalities, ensuring smooth and scalable connectivity.

The development of video game software involves multiple interconnected components (see Figure 1):

- **Specific Low-level Components:** External Tools and Libraries support various tasks such as data parsing, format conversion, and additional

functionalities that enhance development efficiency. These tools help integrate different technologies and optimize workflows.

- **Game Engines:** serve as the foundation for game development, offering reusable components and tools that streamline processes, expand game architecture, and manage assets such as 3D models, textures, and sounds. These engines, whether proprietary or commercial (e.g., Unity [19] and Unreal Engine [9]), code not directly created by the developers.
- **Video Game Content:** Source code represents the custom logic written by the developers to implement game mechanics and user interactions. Developers can create content by writing code directly or using software models, which offer a higher level of abstraction and focus more on content creation rather than implementation details. Commercial engines like Unity and Unreal provide modeling languages (Blueprints in case of Unreal and Unity Visual Scripting in case of Unity), facilitating Model-Driven Game Development [22].

Model-Driven Engineering

Model-Driven Engineering (MDE) [11] is a software engineering paradigm that focuses on the use of models as the primary artifacts throughout the development lifecycle. Unlike traditional approaches that prioritize manual coding, MDE raises the level of abstraction by representing systems through models that capture their structure, behavior, and interactions.

Within the scope of MDE, MDD [13] serves as a practical application of its principles, specifically targeting the software development process. MDD emphasizes the use of high-level models to represent and design systems. Instead of focusing directly on writing source code, MDD shifts the development process toward creating and refining models that define the structure, behavior, and functionality of the application. These models are platform-independent, meaning they abstract away the technical details of specific programming languages or platforms, allowing developers to focus on the core logic and design of the system. Once the models are complete, they are transformed into executable code or interpreted at runtime using tools or frameworks.

In video game development, game engines play a crucial role by integrating various components such as graphics and physics engines, along with a set of tools needed for development. Although widely used game engines such as Unity and Unreal Engine are common choices, some game studios prefer to create their own custom engines designed to meet their specific requirements.

Game engines offer developers a choice between coding, often in languages such as C++, or using software models integrated into the engines. A recent literature review [23] identifies popular engines such as Unity and Unreal, as well as proprietary engines explicitly developed by video game studios.

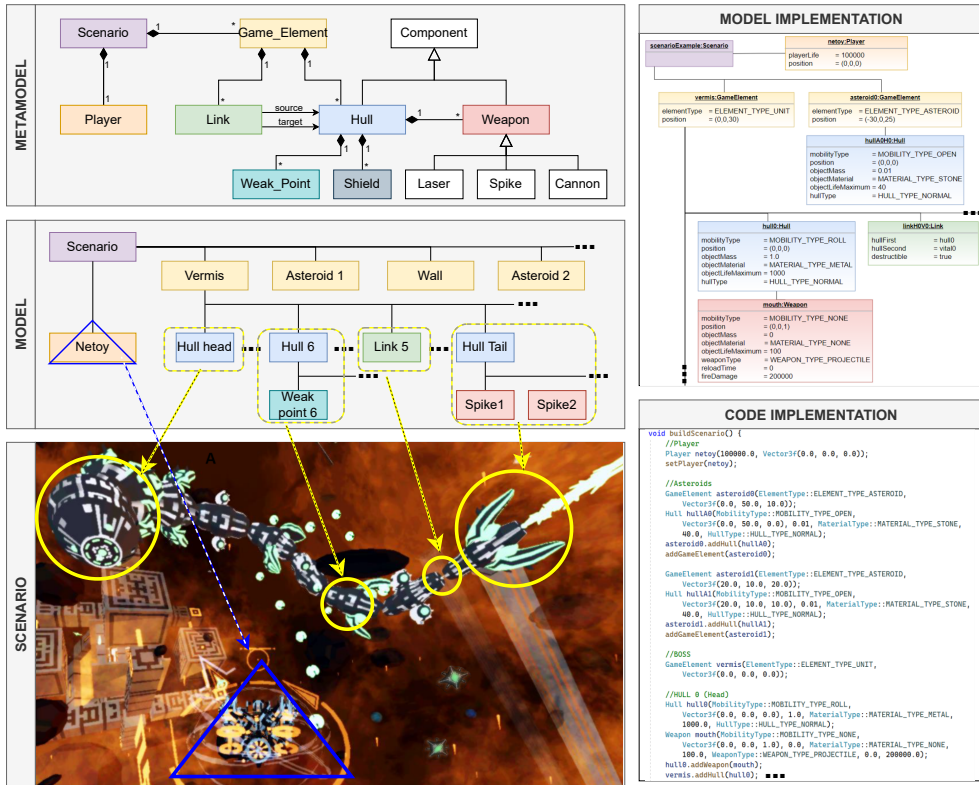
A key component of game engines, whether popular engines or proprietary engines, is their ability to incorporate software models. These models differ from graphical 3D models and serve as high-level abstractions that define game content and interactions. For example, the Shooter Definition Modeling Language (SDML) [4] was developed to create entities in the Kromaia² game, such as bosses, enemies, and environmental elements, as can be seen in Figure 2. SDML models are translated into C++ objects during runtime, allowing developers to design and manage game content more flexibly and efficiently. Despite their advantages, software models in video games are not immune to bugs.

Motivation

The field of GSE is attracting more attention, but research remains limited. Initially compared to traditional Software Engineering in the early 21st century [12], GSE has seen a growing interest over time [1]. However, literature reviews have pointed out significant gaps: few studies focus on case studies and a notable lack of experimental research compared to traditional Software Engineering.

Since those early reviews, the GSE research community has grown substantially, producing over four times more studies, as noted in [5]. Despite this growth, research is still dispersed across multiple venues, making it difficult to consolidate findings. While the field is broadening its focus to include topics like architecture and design and is adopting more advanced empirical methods, many studies still lack thorough critical examination. This Systematic Literature Review [5] identified 13 potential future research directions, one of which is *Testing and Debugging in the Face of Gameplay Subjectivity*. GSE must contend with large codebases, high interactivity among game objects, and vast amounts of assets, making bug reproduction particularly challenging. Moreover, gameplay testing often depends on human testers who must identify bugs while playing, introducing subjective judgments into the process. This underscores the urgent need for automated and innovative testing and debugging techniques to enhance reliability and efficiency in game development.

²Kromaia official trailer released by Playstation: <https://youtu.be/EhsejJBp8Go>



In addition, the adoption of MDD in video games is increasing and has been useful in making video game development easier. However, applying MDE to video games requires specific methods for bug localization to ensure the quality of the software. Currently, most bug localization methods focus mainly on the source code and do not consider other important parts, such as software models. A survey on bug localization [21] shows that none of the existing methods considers models as a source of bugs.

The significance of thorough testing and bug localization in video games cannot be overstated, particularly given their interactive and emergent nature. Bugs can arise from complex interactions between game mechanics, player behavior, and environmental elements, leading to unpredictable outcomes. Furthermore, game developers face unique challenges [14], including managing artifacts such as shaders and meshes, balancing interactive gameplay, and dealing with many possible scenarios. This complexity requires specialized approaches for testing and debugging, particularly in the context of MDD, where bug localization must take into account both the model and its runtime behavior.

Addressing bug localization in models is essential for reducing development time, avoiding delivery delays, and minimizing the risk of releasing bug-ridden games. High-profile cases like *Cyberpunk 2077*³, plagued by numerous bugs at launch, highlights the significant repercussions of inadequate testing and bug resolution in video games.

This thesis is motivated by the critical role of software models in video game development and the need for bug localization techniques tailored to these models.

Search-Based Software Engineering

Given the combinatorial complexity inherent in video game execution, resulting from the wide range of possible player actions, system states, and dynamic interactions, we consider that Search-Based Software Engineering (SBSE) offers a promising approach to address the challenge of bug localization in this domain.

SBSE [10] is an approach that applies optimization techniques, often derived from artificial intelligence and operations research, to solve complex problems in software engineering. By formulating software engineering tasks as search problems, SBSE leverages methods such as genetic algorithms, simulated an-

³How buggy is Cyberpunk 2077, really? - <https://www.pcgamer.com/how-buggy-is-cyberpunk-2077-really/>

nealing, and particle swarm optimization to explore large solution spaces efficiently and find optimal or near-optimal solutions. These techniques are particularly useful in scenarios where the problem space is too vast or complex to address manually.

Research on using SBSE for MDE activities is growing [7]. At the intersection of these fields, model maintenance tasks, such as bug localization, traceability link recovery, and feature location, are reformulated as search problems, requiring the identification of specific model fragments [2, 15, 6, 8]. These efforts leverage search-based optimization techniques, primarily from evolutionary computation, to automate the search for optimal or near-optimal solutions.

Objectives

Given the background described, the objective of this work is to improve bug localization in GSE by exploring successful approaches used in software development and adapting them for application in video game development. To guide this research, we define the following questions:

RQ₁ *What is the current performance of SBSE approaches for bug localization in MDD?*

This research question allows us to study effective approaches to bug localization in MDD, analyzing how different search parameters and model characteristics influence performance. By understanding these impacts, we can optimize SBSE techniques to improve performance in maintaining tasks.

RQ₂ *How do MDD and CcD compare in terms of performance, productivity, and user satisfaction for bug localization tasks in video game development?*

This research question examines the effectiveness of bug localization in MDD and Code-centric Development (CcD) in video game development. The goal is to analyze how each approach affects key factors such as performance, productivity, and user satisfaction when locating bugs. By comparing these paradigms, we aim to identify the strengths and limitations of each method.

RQ₃ *Does considering model interactions improve the performance of existing bug localization techniques in video game development?*

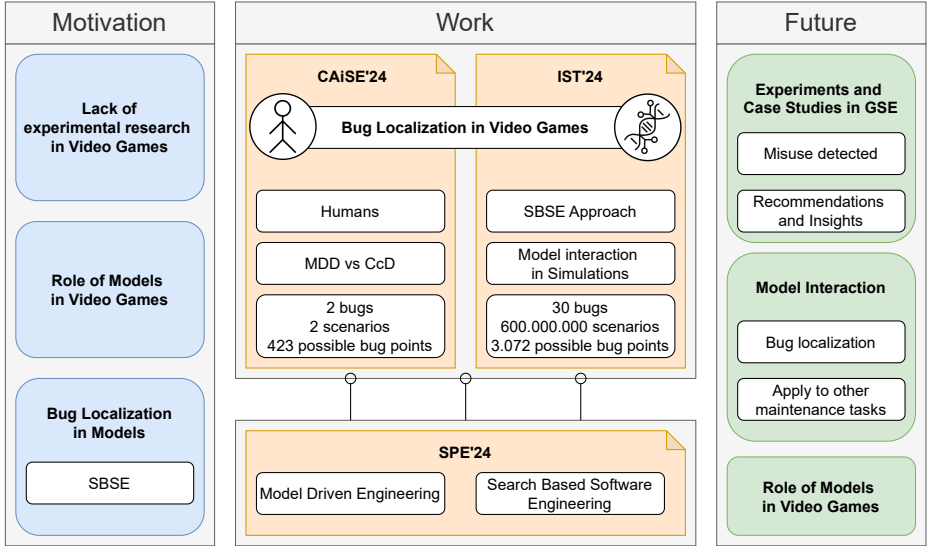


Figure 3: Motivation, work, and future of this thesis.

This research question aims to improve the bug localization approaches in video game development by taking into account model interactions. By analyzing how these interactions influence the performance of bug localization, we aim to determine whether considering model interactions can lead to more accurate and efficient bug localization strategies.

Research Development Progress

Figure 3 summarises the work carried out in this thesis including the motivation, the work done, and the future.

As previously explained, this thesis is motivated by the crucial role that software models play in video game development and the need for effective bug localization techniques specifically designed for these models (left part of Figure 3).

The work presented in this thesis is divided into three main steps (central part of Figure 3), each corresponding to a publication. The following sections describe these steps in detail.

SPE'24 - Study Bug Localization techniques in MDD.

The starting point of this work (bottom of the central part of Figure 3) was the study of the research contexts of MDD and SBSE exploring effective approaches for bug localization. This provided the groundwork for the research carried out next.

SBSE has demonstrated significant effectiveness in tackling problems that can be reformulated as search problems [8, 3]. This makes SBSE a highly promising approach for addressing the challenges of bug localization in MDD. Using SBSE techniques, it is possible to automate bug localization within software models, a task that traditionally relies on manual effort or methods focused solely on source code.

CAiSE'24 - Study the effectiveness of Bug Localization in MDD and CcD in Video Games

Video game development often involves both MDD and CcD approaches, with MDD gaining popularity due to its ability to abstract complex implementation details. However, the effectiveness of bug localization in MDD environments has not been thoroughly explored, particularly in comparison to CcD. Bug localization is a critical task in software development, yet no empirical research has compared MDD and CcD specifically from the perspective of bug location in video game development. Our goal is to analyze how these two approaches impact performance, productivity, and user satisfaction when applied to bug localization tasks (left of the central part of Figure 3). The experiment consists of two bug localization tasks performed by humans, each one in one possible scenario, carried out in both approaches. These two tasks (one scenario each) had 423 possible points where bugs could be found.

IST'24 - Propose an approach to bug localization in GSE that improves on existing ones

Previous work, *CAiSE'24*, highlighted the importance of addressing interactions between models in video games, a topic that, to date, has not been adequately covered in the existing literature. Addressing this, we propose to explore new solutions that incorporate the interactions between software models during simulations (right of the central part of Figure 3). The proposed approach locates 30 bugs, that had 3.072 possible points where bugs could be found, by evaluating 600.000.000 possible scenarios.

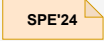
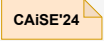
Challenge	Bug Localization Techniques in Models	Bug Localization in Models in Video Games	
Research Questions	RQ1: Bug Localization in Models	RQ2: Bug Localization (models vs code) in video games	RQ3: Our Bug Localization Approach
Publications			
Funded research projects	VARIATIVA: Spanish National R+D+i Plan - PID2021-28695OBI00		
Industrial partners	BSH: Home Appliances Group. Induction hob firmware variability extraction and management tool.	KROMAIA: Modeling, code generation, and evolution for a commercial video game released on PlayStation 4 and Steam.	
	CAF: Variability modeling, code generation, and evolution for railway system software.		

Figure 4: Overview of our contributions.

Finally, future work (right part of Figure 3) focuses on three lines: first, to improve the empirical foundations of GSE; second, to explore and deepen model interaction, not only to improve bug localization but also in other maintenance tasks of video game software; and third, to investigate why models are particularly successful in game development environments, with special attention to hybrid approaches such as Unreal’s Blueprints, which challenge the traditional model-to-code perspective of MDD.

Thesis Overview

Figure 4 shows an overview of the work performed as part of this thesis. It is structured in five different rows: row 1 identifies the challenge that is addressed; row 2 shows the research questions about the challenge; row 3 lists the scientific publications generated; row 4 lists the research projects where the work has been contributed to; row 5 lists the industrial partners where the solutions have been evaluated.

For the first challenge (Bug Localization Techniques in Models), one research question is identified (RQ₁) and one publication is presented, *SPE’24* [18]. For the second challenge (Bug Localization in Models in Video Games), two research questions are identified (RQ₁ and RQ₂), and two publications are presented for each one, *CAiSE’24* [17] and *IST’24* [16], respectively.

The work presented in this dissertation has made contributions to VARIATIVA project: a Spanish national research project whose objective is the use of SBSE techniques to achieve the adoption of SPL approaches.

This thesis was evaluated with the collaboration of three industrial partners: BSH, Europe's leading home appliance manufacturer, with whom we developed a variability extraction and management tool for induction hob firmware; CAF, a global provider of railway solutions, with whom we created a software variability management solution for railway systems; and KROMAIA, a commercial video game available on PlayStation 4 and Steam, which uses its own Modeling Language to create in-game entities.

Research Methodology

This thesis adopts the methodology proposed by R. Wieringa in Design science Methodology for Information Systems and Software Engineering [20], which identifies this research as design science research, since its primary goal is to solve a design problem. The methodology evolves around iterative design research cycles, as illustrated in Figure 5, and consists of the following phases:

- 1. Problem Investigation:** This phase focuses on identifying the core research problems and formulating the corresponding research questions.
- 2. Research Design:** During this phase, approaches are developed to address the research questions effectively.
- 3. Design Validation:** The validity of the proposed approaches is assessed using widely accepted measurements and methods from the relevant literature. Additionally, potential threats to validity are analyzed, and strategies to mitigate them are established.
- 4. Research Execution:** This phase involves implementing the designed approaches and applying them to case studies. The results obtained are documented in a series of research articles.
- 5. Results Evaluation:** Finally, the results are analyzed to answer the research questions and to identify new opportunities for further investigation.

The iterative nature of this process allows the research to evolve dynamically. Starting with the initial research challenge, the knowledge gained and results obtained in each cycle provide new insights, steering the research towards

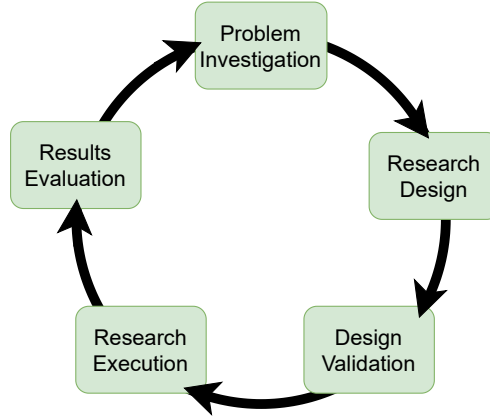


Figure 5: Empirical cycle used in the Design Science methodology.

emerging perspectives. The iterative cycle ensures continuous refinement and adaptation of the research process.

Thesis Structure

This thesis is presented as a compilation of research articles, following the guidelines and regulations for preparing PhD theses established by the Universitat Politècnica de València. According to these guidelines, a thesis structured as a compendium of articles consists of four main sections:

- I. Introduction:** The first section of the thesis (Part I) outlines the research context and motivation, defines the objectives of the study, details the methodology employed throughout the investigation, and provides a summary of the articles included in the compilation.
- II. Compendium of Articles:** The second section (Part II) comprises the set of articles produced during the research. All these articles were published in specialized journals and are presented in the thesis format, preserving their original content and bibliographic references.
- III. Discussion:** The third section (Part III) discusses the results obtained through the research. It also highlights potential future research directions.

IV. Conclusions: The final section of the thesis (Part IV) outlines the conclusions of the study, reflecting on the initial objectives and the results achieved throughout the research.

Compendium Description

The following articles have been produced and published in different journals in the course of the research process presented in this thesis:

1. **On the Interaction between the Search Parameters and the Nature of the Search Problems in Search-Based Model-Driven Engineering** (Software: Practice and Experience) (Roca, Font, Arcega, Cetina, 2024) (**SPE'24**) [18].

This work explores the impact of search parameter values on the performance of evolutionary algorithms in the context of MDE, specifically for Model Fragment Location Problems (MFLPs). It examines how varying parameters such as population size, replacement percentage, and crossover rate influence algorithm performance, along with the role of problem characteristics. Through an evaluation involving 1,895 MFLPs across two industrial case studies and 625 parameter combinations, the study reveals that search parameters significantly affect precision, while problem nature influences recall. The findings underscore the importance of both parameter tuning and understanding problem characteristics for optimizing search-based approaches in SBMDE.

2. **Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games** (International Conference on Advanced Information Systems Engineering) (Roca, Domingo, Pastor, Cetina, Arcega, 2024) (**CAiSE'24**) [17].

This article explores the impact of CcD and MDD on bug localization within video game development. By analyzing a commercial video game released on PlayStation 4 and Steam, the study compares the two approaches in terms of performance, productivity, and user satisfaction. The results revealed that bug localization using MDD increased user satisfaction. However, performance and productivity varied based on experience, with professionals achieving better results using CcD. This suggests that bug localization in MDD is less effective, highlighting a gap in knowledge for handling bugs in MDD environments. As MDD becomes more popu-

lar in video game development, it is important to explore new methods for improving bug localization in these systems.

3. Co-evolving Scenarios and Simulated Players to Locate Bugs that arise from the Interaction of Software Models of Video Games (Information and Software Technology) (Roca, Pastor, Cetina, Arcega, 2024) (**IST'24**) [16].

This work addresses the challenge of bug localization in GSE, particularly within the complex software models used by game engines to represent game elements and their interactions. It introduces CoEBA, a co-evolutionary approach leveraging SBSE techniques to locate bugs in software models, emphasizing interactions between elements like enemies, bosses, or environments. The approach was evaluated using the commercial video game Kromaia, outperforming the developers' baseline method and a random search in terms of precision, recall, and F-measure. Statistical analysis confirmed the significance of these results, demonstrating CoEBA's effectiveness in bug localization in video game development.

Additionally, the three papers included in the compendium were presented in relevant national and international conferences as follows:

- *SPE'24* was presented in the Spanish Conference on Software Engineering and Databases (JISBD 2024), in the context of the Computer Science Spanish Conference (CEDI).
- *CAiSE'24* was presented in the International Conference on Advanced Information Systems Engineering (CAiSE 2024), and in the Video Game Spanish Conference (CEV'24), in the context of the Computer Science Spanish Conference (CEDI). This work was awarded with the Distinguished Paper Award in CAiSE 2024.
- *IST'24* was presented in the Spanish Conference on Software Engineering and Databases (JISBD 2024), in the context of the Computer Science Spanish Conference (CEDI).

Research Support

The research works included as a compendium in this thesis [16, 17, 18] was supported in part by Spanish Ministry of Economy and Competitiveness (MINECO) under the Project VARIATIVA (PID2021-28695OBI00), and in part by the Gobierno de Aragón (Spain) (Research Group T61_23R).

Bibliography

- [1] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2010.05.004> (cit. on pp. 4, 7).
- [2] Lorena Arcega et al. “An approach for bug localization in models using two levels: model and metamodel”. In: *Software and Systems Modeling* 18.6 (2019), pp. 3551–3576. DOI: 10.1007/s10270-019-00727-y (cit. on p. 10).
- [3] Lorena Arcega et al. “Bug Localization in Model-Based Systems in the Wild”. In: *ACM Trans. Softw. Eng. Methodol.* 31.1 (2021). ISSN: 1049-331X. DOI: 10.1145/3472616 (cit. on p. 12).
- [4] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *J. Syst. Softw.* 171 (2021), p. 110804. DOI: 10.1016/j.jss.2020.110804 (cit. on p. 7).
- [5] Jorge Chueca et al. “Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development”. In: *Journal of Systems and Software* 205 (2023), p. 111824. ISSN: 0164-1212 (cit. on p. 7).
- [6] Jorge Echeverría et al. “Comparison of Search Strategies for Feature Location in Software Models”. In: *J. Syst. Softw.* 181.C (2021). ISSN: 0164-1212. DOI: 10.1016/j.jss.2021.111037 (cit. on p. 10).

- [7] Raihana Ferdous et al. “Search-Based Automated Play Testing of Computer Games: A Model-Based Approach”. In: *Search-Based Software Engineering - 13th International Symposium, SSBSE 2021, Bari, Italy, October 11-12, 2021, Proceedings*. Ed. by Una-May O’Reilly and Xavier Devroey. Vol. 12914. Lecture Notes in Computer Science. Springer, 2021, pp. 56–71. DOI: 10.1007/978-3-030-88106-1_5 (cit. on p. 10).
- [8] Jaime Font et al. “Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility”. In: *Software and Systems Modeling* (2021). ISSN: 1619-1374. DOI: 10.1007/s10270-021-00870-5 (cit. on pp. 10, 12).
- [9] Epic Games. *Unreal Engine: The most powerful real-time 3D creation tool*. <https://www.unrealengine.com>. [Online; accessed 21-November-2021]. 1998 (cit. on p. 6).
- [10] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Information and software Technology* 43.14 (2001), pp. 833–839 (cit. on p. 9).
- [11] Stuart Kent. “Model Driven Engineering”. In: *Integrated Formal Methods*. Ed. by Michael Butler, Luigia Petre, and Kaisa Sere. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 286–298. ISBN: 978-3-540-47884-3 (cit. on p. 6).
- [12] Michael McShaffry. *Game Coding Complete*. Paraglyph Publishing, 2003. ISBN: 1932111751 (cit. on p. 7).
- [13] Francisco Durán Muñoz, Javier Troya Castilla, and Antonio Vallecillo Moreno. *Desarrollo de software dirigido por modelos*. 2019 (cit. on p. 6).
- [14] Luca Pascarella et al. “How Is Video Game Development Different from Software Development in Open Source”. In: May 2018. DOI: 10.1145/3196398.3196423 (cit. on p. 9).
- [15] Francisca Pérez et al. “Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study”. In: *Inf. Softw. Technol.* 103 (2018), pp. 188–201 (cit. on p. 10).

-
- [16] Isis Roca et al. “Co-evolving scenarios and simulated players to locate bugs that arise from the interaction of software models of video games”. In: *Information and Software Technology* 169 (2024), p. 107412. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2024.107412> (cit. on pp. 13, 17, 18).
- [17] Isis Roca et al. “Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games”. In: *Advanced Information Systems Engineering*. Ed. by Giancarlo Guizzardi et al. Cham: Springer Nature Switzerland, 2024, pp. 581–595. ISBN: 978-3-031-61057-8 (cit. on pp. 13, 16, 18).
- [18] Isis Roca et al. “On the interaction between the search parameters and the nature of the search problems in search-based model-driven engineering”. In: *Software: Practice and Experience* 54.8 (2024), pp. 1312–1336. ISSN: 0038-0644. DOI: <https://doi.org/10.1002/spe.3320>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3320> (cit. on pp. 13, 16, 18).
- [19] Unity Technologies. *Unity Real-Time Development Platform / 3D, 2D VR & AR Engine*. <https://unity.com>. [Online; accessed 21-November-2021]. 2005 (cit. on p. 6).
- [20] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014 (cit. on p. 14).
- [21] W. Eric Wong et al. “A Survey on Software Fault Localization”. In: *IEEE Trans. Software Eng.* 42.8 (2016), pp. 707–740. DOI: 10.1109/TSE.2016.2521368 (cit. on p. 9).
- [22] Meng Zhu and Alf Wang. “Model-driven Game Development: A Literature Review”. In: *ACM Computing Surveys* 52.6 (2019), pp. 1–32. ISSN: 0360-0300. DOI: 10.1145/3365000 (cit. on p. 6).
- [23] Meng Zhu and Alf Inge Wang. “Model-Driven Game Development: A Literature Review”. In: *ACM Comput. Surv.* 52.6 (2019). ISSN: 0360-0300. DOI: 10.1145/3365000 (cit. on p. 7).

Part II

Compendium of Articles

On the Interaction between the Search Parameters and the Nature of the Search Problems in Search-Based Model-Driven Engineering

The use of Search-Based software engineering to address Model-Driven Engineering activities (SBMDE) is becoming more popular. Many maintenance tasks can be reformulated as a search problem, and, when those tasks are applied to software models, the search strategy has to retrieve a model fragment. There are no studies on the influence of the search parameters when applied to software models. This paper evaluates the impact of different search parameter values on the performance of an evolutionary algorithm whose population is in the form of software models. Our study takes into account the nature of the model fragment location problems (MFLPs) in which the evolutionary algorithm is applied. The evaluation searches 1,895 MFLPs (characterized through five measures that define MFLPs) from two industrial case studies and uses 625 different combinations of search parameter values. The results show that the impact on the performance when varying the population size, the replacement percentage, or the crossover rate produces changes of around 30% in performance. With regard to the nature of the problems, the size of the search space has the largest impact. Search parameter values and the nature of the MFLPs influence the performance when applying an evolutionary algorithm to perform fragment location on models. Search parameter values have a greater effect on precision values, and the nature of the model fragment location problems has a greater effect on recall values. Our results should raise awareness of the relevance of the search parameters and the nature of the problems for the SBMDE community.

Author version of the article: I. Roca, J. Font, L. Arcega, and C. Cetina, “On the Interaction between the Search Parameters and the Nature of the Search Problems in Search-Based Model-Driven Engineering”. In: Software: Practice and Experience, Volume 54, Issue 8, Pages 1312-1336, doi: 10.1002/spe.3320.

1.1 Introduction

There is a growing body of research on the use of Search-Based Software Engineering (SBSE) to address Model-Driven Engineering (MDE) activities [5]. In the intersection between MDE [19] and SBSE [18], activities related to model maintenance are reformulated as search problems. Specifically, tasks such as bug localization, traceability link recovery, or feature location can be reformulated as search problems where a model fragment must be found [1, 27, 8]. These works use search-based optimization techniques (mainly those from the evolutionary computation literature) to automate the search for optimal and near-optimal solutions.

In this work, we apply Search-Based Model-Driven Engineering (SBMDE) to address location problems in models. This activity aims to identify the parts of the models that are relevant to a specific task, and the result comes in the form of a model fragment. Model fragment location is one of the most important search problems in models. For example, in a model with 500 elements, the number of model fragments that can be generated can reach the value of 10^{29} [13]. Since the search space is so large, it is not practical to thoroughly explore the space of possibilities.

In our previous works [12, 16], we proposed an approach for locating model fragments using an evolutionary algorithm. To achieve this, the evolutionary algorithm keeps a population of candidate solutions (in the form of model fragments) and evolves them using genetic operators that are designed to work with model fragments.

However, in the current SBMDE practice, key search parameter values (population size, replacements, mutation rate, and crossover rate) are selected by conventions and ad hoc choices. Nevertheless, the default values used by the SBMDE community [5] are borrowed from other domains, and there are no specific studies about what search parameter values should be used when working with software models.

In addition, works on SBMDE provide many details about the techniques that are used to perform the search, however, there is a lack of detail about the nature of the problems used. The proper reporting of the nature of the problems used is important so that the results obtained can be useful to other practitioners. Therefore, we characterize the nature of problems by means of five measures that define model fragment location problems (size, volume, density, multiplicity, and dispersion) [4].

In this work, we perform an evaluation to determine which search parameter has the highest impact on the performance of the search based on the nature of the location problems. To do so, we apply the search strategy to 1,895 Model Fragment Location Problems (MFLP) from industrial case studies using different sets of search parameter values (625 different combinations). Then, we calculate the performance by means of precision, recall, and F-measure values [23], comparing the results provided in each case with the oracle extracted from the case studies (considered the ground truth). Finally, we perform a statistical analysis of the results obtained in order to determine whether the different search parameter values applied and the nature of the MFLPs have an impact on the performance or whether the differences are obtained by mere chance.

The results show that the choice of good search parameters can provide a boost in performance (precision mean values range from 6.48% for non-optimal parameters to values up to 73.08% for optimal parameters, while mean recall values range from 62.02% up to 75.45%). Moreover, when varying the population size, replacement percentage, or crossover rate parameters, the differences in performance are around 30%, but, when varying the mutation rate parameter, the differences in performance remain below 5%. Our statistical analysis shows that the crossover rate parameter has the largest impact on our case studies on industrial models. In addition, the nature of the model fragment location problems also influences the precision and recall values. Low values in the measures related to the search space obtain better values in performance, while high values in measures related to the model fragment outperform the low values. In this case, the size of the search space has the greatest impact. It is important to highlight that while the search parameter values have a greater impact on precision values, the nature of the MFLPs has a greater impact on recall values.

The evaluation presented in this paper is an initial work on the optimization that can be achieved when applying an evolutionary algorithm to perform fragment location on models. We want to advise the SBMDE community that they could be missing out on the relevance of the search parameters and the

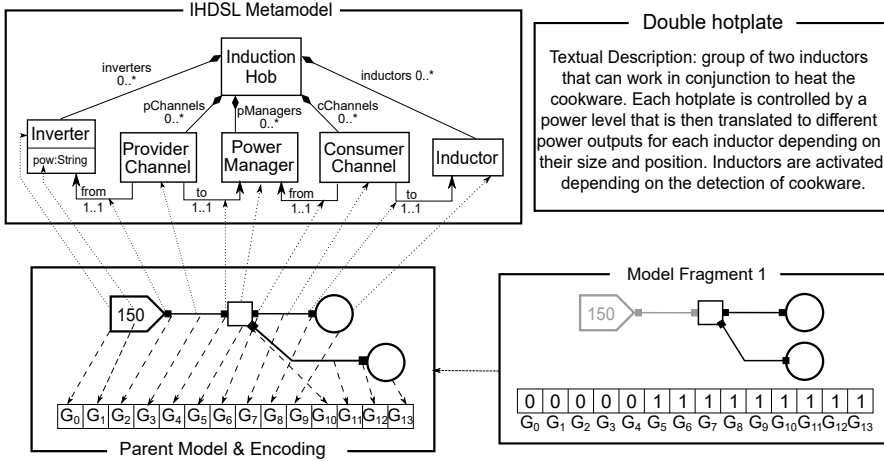


Figure 1.1: IHDSL metamodel, product model, and model fragment.

nature of their case studies for SBMDE approaches. We present this evaluation with the hope that it can be useful for practitioners from the SBMDE community when looking for default values or advice on how to balance their search strategy to boost performance.

The rest of the paper is organized as follows. Section 1.2 presents some background about model fragment location using an evolutionary algorithm. Section 1.3 shows the process that is followed to evaluate the impact of the different parameter values taking into account the measures for reporting model fragment location problems. Section 1.4 presents and discusses the results of the evaluation. Section 1.5 examines some related work. Finally, we conclude the paper in Section 1.6.

1.2 Model fragment location using an evolutionary algorithm

In this section, we present the model fragment location problem based on the products of one of our industrial partners, the search strategy used to address the model fragment location problems, the measures that define model fragment location problems, and the search parameters of the evolutionary algorithm.

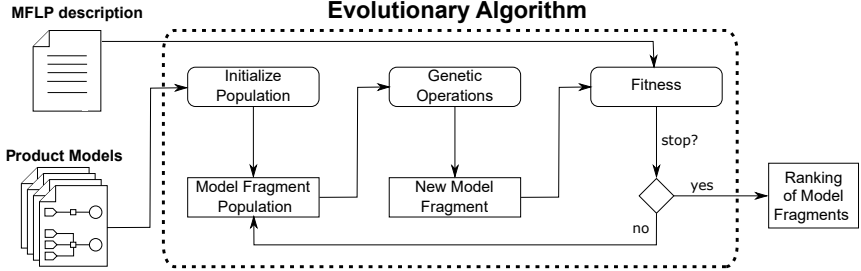


Figure 1.2: Overview of the search strategy.

1.2.1 Model fragment location problem (MFLP)

Figure 1.1 shows the Domain-Specific Language (DSL) used by one of our industrial partners, BSH, to formalize its products. This DSL is used to describe the models of the induction hobs that will be part of the evaluation. The firmware of the BSH products is generated from the DSL models. The DSL used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 metaclasses, 47 relations with each other, and more than 180 properties. For legibility reasons and due to intellectual property rights concerns, in this section, we show a simplified subset of the IHDSL (see Figure 1.1, IHDSL Metamodel). However, the evaluation was performed using the full IHDSL that is used in BSH.

The Parent Model in Figure 1.1 depicts an example of a model that is specified with the IHDSL. In this example, the induction hob aggregates a 150-power inverter that is connected to a single power manager through a provider channel. The power manager is connected to a double inductor through two consumer channels. The dotted lines show the metamodel concepts that are related to the model. The bottom part shows the encoding of the model. This encoding will be used to define model fragments on the parent model, which is explained in the next subsection.

1.2.2 Search strategy: an evolutionary algorithm for MFLP

To develop an evolutionary algorithm, the following elements must be defined: 1) an encoding of the problem that can be used to represent the individuals; 2) a fitness function that can be used to evaluate how good each individual is as a solution to the problem; and 3) a set of genetic operators that can be applied to modify and evolve the population of individuals.

Figure 1.2 shows an overview of the evolutionary algorithm used. The left part shows the inputs for the approach: a MFLP description and a set of product models. The center 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. The 'Genetic Operations' step produces a new generation of model fragments. Finally, the 'Fitness' step assigns values that assess how good each model fragment is based on the description. As output, the approach provides a list of model fragments that might be relevant as a solution for the location problem.

Encoding

In this case, the candidates are model fragments. To represent a candidate solution (individual), we use a binary string. Each position of the binary string represents a model element (concept, properties, or relationships) related to a parent model. Thus, a candidate solution is defined as a set of model elements that are associated with a parent model. The size of the solution represents the number of model elements in the string. This representation defines model fragments, indicating the presence or absence of each of the model elements of the parent.

Model Fragment 1 in Figure 1.1 shows a model fragment that was defined in the Parent Model. All of the model fragments are defined concerning a parent model. When the position is equal to 0, it indicates that the element is not part of the model fragment. However, when the position is equal to 1, it indicates that the element is part of the model fragment. Thus, we can indicate the model elements from the model parent that compose the model fragment. For instance, Model Fragment 1 is a model fragment that contains the power manager, the two consumer channels, and the two inductors of the parent model.

Fitness function

The search strategy is guided using a fitness function that is based on textual comparisons [16]. In this case, the algorithm assesses the relevance of each model fragment in relation to the description provided by the software engineers of our industrial partners (see the upper-right part of Figure 1.1).

The description uses natural language. To assess the relevance of each model fragment to the description, we apply a technique that is based on information retrieval. First, the text from the textual description and the models is homogenized through the use of natural language processing techniques [22]. Then, we apply Latent Semantic Indexing (LSI) [20] to analyze the relationship between the description and the solution candidates generated. The result is a ranking of model fragments that are ordered based on the similitude with the description.

Genetic operators

The genetic manipulation of the individuals is performed using four different operators. These operators were defined to work on model fragments [15, 16]:

- **Parent Selection:** This operator is responsible for selecting the parent individuals that will serve as the base for the new offspring. In this approach, we use a generic roulette wheel selection, which assigns a probability of being selected that is proportional to each individual's fitness value. All individuals can be selected as parents, but the higher the fitness value, the higher the probability of being selected as a parent for the next generation.
- **Crossover:** The crossover operator aims to combine the genetic material from the parents into new individuals. In this approach, we use a multiple-point crossover operator that is based on a mask [15] that will combine two model fragments into two new individuals. The mask determines how the combination is done. For each element of the model fragments, it indicates if the offspring should inherit from one parent or the other (including/excluding the element depending on whether or not the element is present in the parent). As a result, two individuals are generated, one by applying the mask directly and another one by applying the inverse of the mask. The crossover operator is not always applied to the new offspring; the crossover rate parameter (p_c) determines whether or not it is applied.

- **Mutation:** The mutation operator aims to imitate the spontaneous mutations that occur in nature. The mutation can turn into an advantage or a disadvantage for the survival of the individual. In this approach, we use an evenly distributed mutation across the genes of the individual. The mutation operator can perform two kinds of modifications: the addition of elements to the fragment (by inverting a 0 for a 1 in the binary string), or the removal of elements from the model fragment (by inverting a 1 for a 0 in the binary string). Again, the mutation is not always applied to the new offspring; the mutation rate parameter (p_m) determines whether or not it is applied.
- **Replacement:** The replacement operator aims to modify the current population by combining it with the new offspring generated with previous operators. In this approach, we replace the less fit part of the current population with new offspring. Two parameters will determine the outcome of the operator: the population size (μ), which will be kept constant throughout the entire execution of the search, and the replacement factor (λ), which determines the number of individuals that are replaced in each generation.

1.2.3 Measures that define model fragment location problems (MD-MFLP)

Some research works [4, 8] use a set of five measures that define and characterize model fragment location problems (MFLP). Two of the five measures characterize the model parent where the MFLP is going to be located (search space), and the other three measures characterize the model fragment that realizes the MFLP (solution). These measures are the following:

Search Space Size (SS-Size) measures the number of elements of the model where the MFLP is located. The elements are concepts, properties, or relationships that are present in the parent model. The SS-Size value of the model presented in Figure 1.1 is 14, which matches the number of bits in the encoding.

Search Space Volume (SS-Volume) measures the number of models that compose the search space. The location of a MFLP may be not limited to a single model and sometimes has to be performed in more than one model. In the example from Figure 1.1, the SS-Volume value is 1 because the search is performed in one model. If it is necessary to locate the same

MFLP in more than one model, one model fragment is obtained for each model used in the location.

Model Fragment Density (MF-Density) measures the percentage of model elements that realize a solution, in other words, the ratio of model elements in the model fragment and the model elements of the parent model. In the example from Figure 1.1, the MF-Density value is 0.64 (9 model elements in the model fragment divided by 14 model elements in the parent model).

Model Fragment Multiplicity (MF-Multiplicity) measures the number of times the solution appears in the search space. In the example from Figure 1.1, the MF-Multiplicity value is 1 because the MFLP only appears once.

Model Fragment Dispersion (MF-Dispersion) measures the ratio of connected elements in the solution. It is computed as the ratio between the number of groups and the number of model elements of the model fragment. In the example from Figure 1.1, the MF-Dispersion value is 0.07 (1 group in the model fragment divided by 14 model elements in the model fragment). This value can range from 0 to 1. Values around 0 indicate a strong connection among the model fragment elements, while values around 1 indicate a strong dispersion among the model fragment elements.

In this work, we characterize the case studies and the results obtained with these specific measures that define model fragment location problems. In addition, we focus on the search parameters from the genetic operators that can be tuned to boost the performance of the approach.

1.2.4 Search parameters of the evolutionary algorithm for model fragment location

Evolutionary algorithm researchers acknowledge that good search parameter values are essential for good evolutionary algorithm performance [9]. In addition, the literature distinguishes between qualitative parameters (finite domain with no sensible distance or ordering) and quantitative parameters (infinite domain with structure or order) [9].

For example, in our approach, we define our own qualitative parameter for the crossover operator parameter, which describes how the crossover will be done (the operator itself). However, since the crossover rate expresses the proba-

Quantitative parameters	Set of values
Population size (μ)	50, 100, 150, 200, 250
Replacements (λ)	20%, 30%, 40%, 50%, 60%
Mutation rate (p_m)	0.2, 0.4, 0.6, 0.8, 1.0
Crossover rate (p_c)	0.2, 0.4, 0.6, 0.8, 1.0

Table 1.1: Representation of quantitative parameters representation.

bility of the crossover operator occurring and can take any value between 0 and 1, it is considered a quantitative parameter. For both types of parameters, the elements of the parameter's domain are called parameter values and we instantiate a parameter by allocating a value to it. In this case study, the qualitative parameters (operators indicated in previous sections) will be fixed from the beginning; we are going to evaluate the search approach using different parameter values for the quantitative parameters.

Table 1.1 shows the most common quantitative search parameters that are used in current practice and will be used to drive our approach. The population size (μ) parameter describes the number of model fragments that are maintained as candidate solutions in each generation. The replacement (λ) parameter indicates the percentage of the population that is going to be discarded in the next iteration. Therefore, it also determines the number of new individuals that will be generated through the application of the genetic operators. The mutation rate (p_m) and the crossover rate (p_c) define the probability of applying the mutation operator or the crossover operator in each case.

A set of values for each one of the parameters is known as a parameter vector. In our case, the parameter vector $p = \{\mu, \lambda, p_m, p_c\}$ can take the values defined in Table 1.1 for each one of the search parameters. For each search parameter, we included both high and low values to properly explore the alternatives. During the evaluation, 625 different parameter vectors are used, so all of the combinations of values are explored. (e.g., $p_1 = \{100, 20\%, 0.2, 0.8\}$, $p_2 = \{150, 10\%, 0.4, 0.6\}$).

The parameters and values taken into account for the experimentation are accepted parameters and values in the literature [3, 32]. Table 1.2 shows the similarities and differences between previous works in parameter tuning: Arcuri and Fraser [3], Sayyad et al. [32], and this work.

There are two atomic performance measures for Evolutionary Algorithms. The first one measures the solution quality (already evaluated in our previous work [14, 15]), and the second one measures the algorithm speed or search effort

Table 1.2: Comparison to other similar works.

Parameter	Arcuri and Fraser [3]	Sayyad et al. [32]	This work
Algorithm	Genetic Algorithm	NSGA-II, IBEA	Genetic Algorithm
Mutation rate	0.1	{0, 0.5, 1, 1.5, 2}/Features	{0.2, 0.4, 0.6, 0.8, 1}
Crossover rate	{0, 0.2, 0.5, 0.8, 1}	{0, 0.2, 0.5, 0.8, 1}	{0.2, 0.4, 0.6, 0.8, 1}
Population size	{4, 10, 50, 100, 200}	{10, 50, 100, 150, 200}	{50, 100, 150, 200, 250}
Elitism rate	{0, 1, 10%, 50%} or steady state	Elitism rate does not apply to either NSGA-II or IBEA. Steady state is implemented in jMetal for NSGA-II but not IBEA. Thus no variation will be considered here.	{20%, 30%, 40%, 50%, 60%}
Selection	roulette wheel, tournament with size either 2 or 7, and rank selection with bias either 1.2 or 1.7	Binary tournament only	roulette wheel
Parent replacement	activated or not	no	activated
Repeats	15	15	100
Testbed	20 classes	20 feature models	1,895 Model Fragments Location Problems (MFLPs)
Number of configurations	$5 \times 5 \times 5 \times 5 \times 2 = 1,250$	$2 \times 5 \times 5 \times 5 = 250$	$5 \times 5 \times 5 \times 5 = 625$ for each MFLP (more than 1M)
Performance measurements	Relative coverage	Correctness and Hypervolume	Precision, recall, and F-measure
Statistical method	Statistical difference (Mann-Whitney U-test) and effect size (Vargha-Delaney $\hat{A}_{12}$)	Statistical difference (Mann-Whitney U-test) and effect size (Vargha-Delaney $\hat{A}_{12}$)	Statistical difference (ANOVA Tukey HSD test) and effect size (Cohen d value)

(evaluated in [12]). In this work, we try to maximize the quality of the solution obtained by the approach by varying the parameter vectors. The approach is executed for a fixed amount of time using one of the parameter vectors. Then, the quality of solutions achieved is compared to determine which parameter vector provides the best quality.

1.3 Experimental setup

This section presents the process followed to evaluate the impact of the different parameter values taking into account the measures that define model fragment location problems.

RQ1 Is there any impact on performance regarding the search parameter vector for model fragment location problems?

RQ2 Is there any impact on performance regarding the measures that define model fragment location problems?

RQ3 Are there any relations between the measures that define the model fragment location problems and the search parameter vector used that affect the performance?

To evaluate the impact of the different parameter values on the results, we execute the approach using a different set of parameter values each time to solve 1,895 MFLP from two of our industrial partners: BSH, the leading manufacturer of home appliances in Europe; and CAF, an international provider of railway solutions worldwide. In addition, we perform a statistical analysis to ensure the validity of the results.

We use the product models from our industrial partners as an oracle to evaluate the results. In other words, we make use of a set of product models whose MFLP realizations are known beforehand and that are considered to be the ground truth, thus allowing us to compare the results provided by our approach with the oracles.

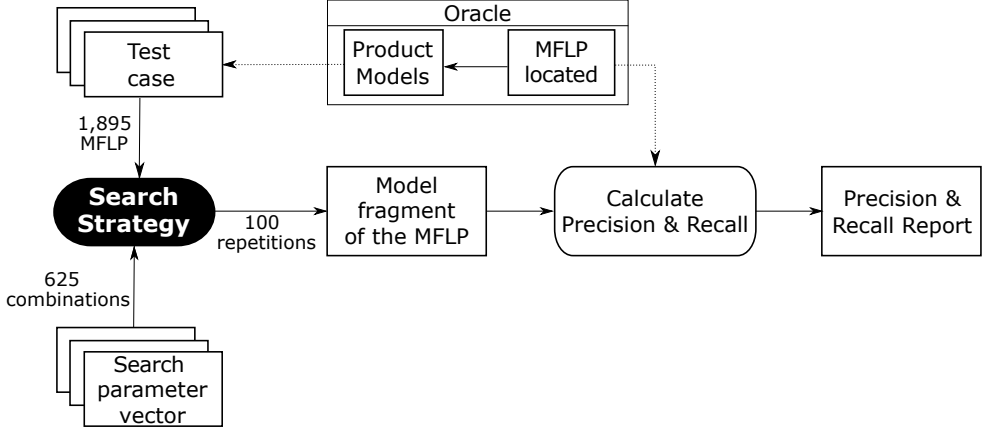


Figure 1.3: Overview of the evaluation with the oracle.

1.3.1 Setup of the case studies

Figure 1.3 shows an overview of the process that we have followed in the evaluation. The top-right part presents the oracle (a set of product models with the MFLP located and formalized).

First, we construct a test case for each MFLP that is present in the oracle. In addition, we generate the 625 different parameter vectors with values from Table 1.1 to configure the algorithm (left part of Figure 1.3). We run each test case for each one of the possible parameter vectors for a fixed time. The allocated time is 10 s (a prior test showed that the search converges in less than the allocated time). This results in a model fragment for each of the test cases for each parameter vector. Finally, the solutions are compared with the model fragments from the oracle (considered the ground truth) in order to obtain the precision, recall, and F-measure values. The operation is repeated 100 times for each combination of the parameter vector and test case to reduce the stochastic component that algorithms of this type have. Finally, the data is aggregated into a report containing the results of the executions (shown in Section 1.4.1).

The Model Fragment Location Problems (MFLPs) used in this evaluation were obtained from two of our industrial partners, BSH and CAF:

BSH

BSH is one of the leading companies in the home appliances sector. We collaborated with the induction division, which has been creating the firmware of Induction Hobs (IHs) for brands like Bosch and Siemens for the last fifteen years. The latest firmware produced includes full cooking surfaces, where heating areas are dynamically created and activated or deactivated depending on the characteristics (size, position, material, etc.) of the cookware placed on top.

There has also been an increase in the feedback that the hob provides to the cook, such as temperatures of the food being cooked in the cookware or even real-time values of the actual consumption of the Induction Hob. In this evaluation, we use 608 MFLPs extracted from the products that they develop. The oracle is composed of the description of each MFLP, the models of the products, and the model fragment that corresponds to each of the MFLP¹.

CAF

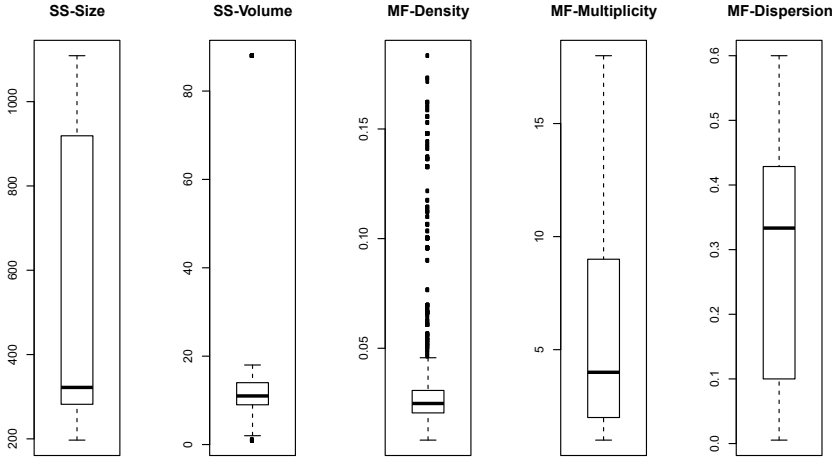
We also use the models from CAF in our evaluation. CAF is a constructor of railway solutions. They produce trains in many different forms (regular trains, subway, light rail, monorail, etc.) that are distributed worldwide. A train includes different pieces of specific equipment to carry out specific tasks for the train. These are located in vehicles and cabins and are usually designed and manufactured by different providers.

The DSL used by CAF can be used to describe the interaction between the different pieces of equipment on the train. Moreover, the DSL allows specifying non-functional aspects that are related to regulations, such as the different levels of redundancy present in the system or the quality of signals from the equipment. This results in a DSL that is composed of around 1,000 different elements.

In this evaluation, we use 1,287 MFLPs extracted from the products that they develop. The oracle is composed of the description of each MFLP, the models of the products, and the model fragment that corresponds to each of the MFLPs².

¹The following video shows the IH models and model fragments used by BSH: www.youtube.com/watch?v=nS2sybEv6j0

²The following video shows the train models and model fragments used by CAF: www.youtube.com/watch?v=Ypc12evEQB8



Measure	Search Space (SS)		Model Fragment (MF)		
	Size	Volume	Density	Multiplicity	Dispersion
Min	197.0	1.00	0.0081	1.000	0.0053
1st Q	282.0	9.00	0.0205	2.000	0.1000
Median	322.0	11.00	0.0248	4.000	0.3333
Mean	515.7	14.48	0.0316	5.993	0.2784
3rd Q	919.0	14.00	0.0308	9.000	0.4286
Max	1,109.0	88.00	0.1834	18.000	0.6000
Sd	327.91	16.63	0.0271	4.569	0.1709

Figure 1.4: Box plots with the value for each MD-MFLP obtained from the 1,895 MFLPs and report of the search problems used in the evaluation.

Therefore, in this evaluation, we have 1,895 different MFLPs provided by our industrial partners. The use of two different domains with a wide variety of MFLPs and their casuistry has led to an improvement in the generalizability of our assessment.

We classified each of the MFLPs by means of the Measures that define Model Fragment Location Problems (MD-MFLP) (Section 1.2.3). For each MD-MFLP, we define two groups: HIGH and LOW. To do this, we use a median-based discretization by splitting each MD-MFLP by the value of the sample median [24, 21]. Figure 1.4 shows the values for each of the five MD-MFLPs of the MFLPs used in this evaluation. Values above the median are considered to be HIGH, while values below the median are considered to be LOW. For instance, the median of the MF-Multiplicity of all of the MFLPs is 4; all MFLPs whose MF-Multiplicity value are above 4 are defined as HIGH for MF-Multiplicity; in contrast, all MFLPs whose MF-Multiplicity value are below 4 are defined as LOW for MF-Multiplicity.

1.3.2 Performance measurements

Once the results from applying the approach to the test cases are obtained, we proceed to compare them with the oracle and measure them in terms of some software quality properties. Figure 1.5 shows an example of a model fragment from the oracle (left), a model fragment candidate obtained from the application of the approach (right), and the confusion matrix [33] used to compare both (middle).

A confusion matrix is a table that is often used to describe the performance of a classification model (our approach under evaluation) on a set of test data (the resulting model fragments) for which the true values are known (from the oracle). In this case, each MFLP realization returned by the approach is a model fragment that is composed of a subset of the model elements that are part of the product model (where the MFLP is located). Since the granularity will be at the level of model elements, the presence or absence of each model element will be considered as a classification. Therefore, our confusion matrices will distinguish between two values: TRUE (presence) and FALSE (absence).

Figure 1.5 shows an example of the comparison process performed to compare a result from one of the evaluated approaches with the ground truth from the oracle and the resulting confusion matrix. The left part shows the actual realization of MFLP #1 (obtained from the oracle and considered the ground truth), while the right part shows the predicted realization of MFLP #1 output

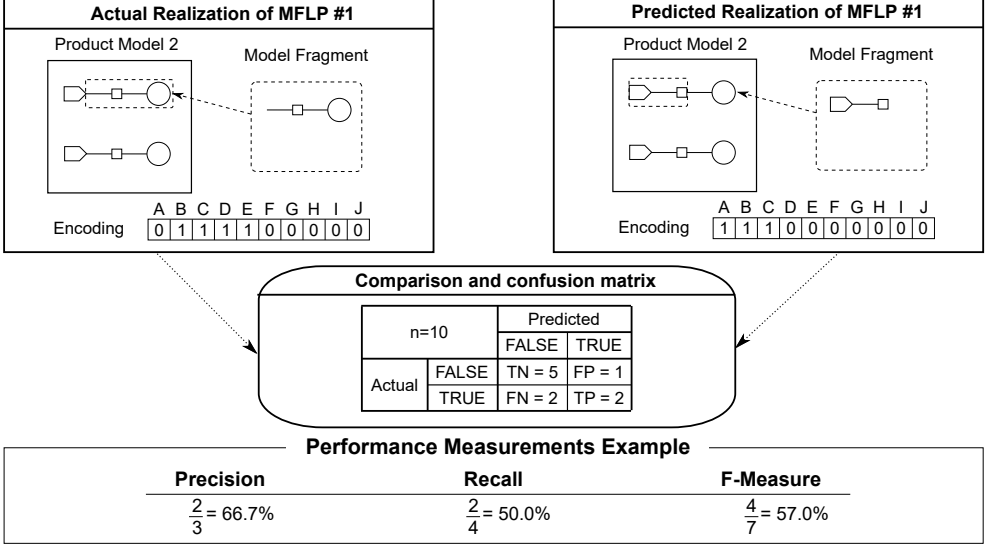


Figure 1.5: Example of a confusion matrix for two model fragments.

by the approach. The confusion matrix arranges the results of the comparison into four categories:

True positive (TP): A model element present in the predicted realization that is also present in the actual realization (e.g., model element B is a TP).

True Negative (TN): A model element not present in the predicted realization that is also not present in the actual realization (e.g., model element H is a TN).

False Positive (FP): A model element present in the predicted realization that is not present in the actual realization (e.g., model element A is an FP).

False Negative (FN): A model element not present in the predicted realization that is present in the actual realization (e.g., model element D is an FN).

The confusion matrix holds the results of the comparison between the predicted results and the actual results; it is just a specific table layout to help the visualization of the performance of a classifier. However, to evaluate the

performance of the approach, it is necessary to derive some measurements from the values of the confusion matrix, which in this case are the three measurements: precision, recall, and F-measure.

Precision measures the number of elements from the prediction (the result of the approach) that are correct according to the ground truth (the oracle).

$$Precision = \frac{TP}{TP + FP} \quad (1.1)$$

Recall measures the number of elements of the ground truth (the oracle) that are correctly retrieved by the prediction (the result of the approach).

$$Recall = \frac{TP}{TP + FN} \quad (1.2)$$

F-measure combines both recall and precision as the harmonic mean of precision and recall.

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (1.3)$$

Precision and recall values can range between 0% to 100%. Following up with the example of the confusion matrix in Figure 1.5, we can calculate the precision, recall, and F-measure for the model fragment (see Figure 1.5). The model fragment has a measurement of 66.7% in precision (two out of the three elements included in the candidate model are present in the model fragment from the oracle) and 50% in recall (2 out of the 4 elements that are present in the oracle are also present in the model fragment). This results in a combined F-measure of 57%.

Details about the implementation of the search strategy approach can be seen in [12, 15]. We performed the execution of the approach using an array of computers with 8 core processors, clock speeds of 4GHz, and 16 GB of RAM. All of them were running Windows 10 Pro N 64 bits as the hosting Operative System and the Java(TM) SE Runtime Environment (build 1.8.0_73-b02).

Since the software models of the case studies are currently operating or will be released in the near future, this information is limited by confidentiality agreements with our industrial partners. Nevertheless, for purposes of replicability, the CSV files with the results of this evaluation and that were used as input in the statistical analysis are published online at: https://svit.usj.es/SPE_Roca_data/.

1.4 Results

This section presents the results obtained in the evaluation and the statistical analysis that answers each research question, the findings, the discussion of the results, and the threats to validity.

1.4.1 Evaluation results

This section is divided into three parts. The first part shows the performance of the four search parameters at each of their five possible values. The second part shows the performance of the MD-MFLP measures for the two levels established (LOW and HIGH). The third part shows the performance taking into account both the search parameters and the measures.

Performance by search parameter value

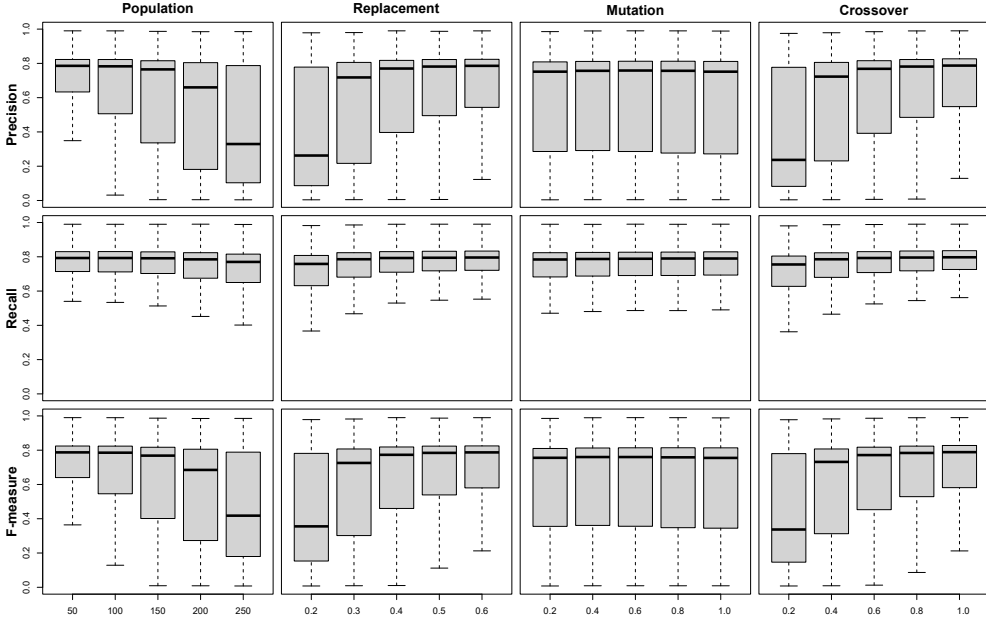
The bottom part of Figure 1.6 shows the mean values and standard deviations that were calculated for each search parameter value. We also show the F-measure value that combines precision and recall into a single performance indicator.

For the population size, the $\mu_1 = 50$ value achieved the best results reaching mean values of 68.45% in precision, 73.26% in recall, and 69.67% in F-measure. However, all of the values provided similar results in terms of recall with differences below 5%.

For the replacement percentage, the $\lambda_5 = 60\%$ value achieved the best results reaching mean values of 66.84% in precision, 73.76% in recall, and 68.46% in F-measure. Again, the differences in terms of recall when using each of the parameter values were small (below 5%).

For the mutation rate, the $p_{m2} = 0.4$ value achieved the best results in terms of precision with a mean of 58.00%, while the $p_{m5} = 1.0$ value achieved the best results in terms of recall with a mean of 73.01%. However, the performance differences when using each parameter value were small (below 5%).

For the crossover rate, the $p_{c5} = 1.0$ value achieved the best results reaching mean values of 67.32% in precision, 74.30% in recall, and 68.85% in F-measure. Again, the differences in terms of recall when using each of the parameter values were small (below 5%).



Parameters		Precision $\pm$ (σ)	Recall $\pm$ (σ)	F-measure $\pm$ (σ)
Population size μ (50-250)	$\mu_1 = 50$	68.45 $\pm$ 21.85	73.26 $\pm$ 16.10	69.67 $\pm$ 20.14
	$\mu_2 = 100$	65.49 $\pm$ 24.98	73.23 $\pm$ 16.04	67.14 $\pm$ 22.81
	$\mu_3 = 150$	59.86 $\pm$ 28.50	72.98 $\pm$ 15.90	62.35 $\pm$ 25.81
	$\mu_4 = 200$	51.90 $\pm$ 31.26	72.33 $\pm$ 15.75	55.40 $\pm$ 28.37
	$\mu_5 = 250$	42.02 $\pm$ 32.06	71.20 $\pm$ 15.62	46.58 $\pm$ 29.43
Replacement percentage λ (20-60)	$\lambda_1 = 20\%$	39.32 $\pm$ 32.44	70.03 $\pm$ 15.75	43.79 $\pm$ 29.99
	$\lambda_2 = 30\%$	54.14 $\pm$ 30.47	72.32 $\pm$ 15.77	57.37 $\pm$ 27.58
	$\lambda_3 = 40\%$	62.03 $\pm$ 26.92	73.28 $\pm$ 15.85	64.32 $\pm$ 24.28
	$\lambda_4 = 50\%$	65.37 $\pm$ 24.69	73.62 $\pm$ 15.90	67.20 $\pm$ 22.32
	$\lambda_5 = 60\%$	66.84 $\pm$ 23.51	73.76 $\pm$ 15.93	68.46 $\pm$ 21.32
Mutation rate ρ_m (0-1)	$\rho_{m1} = 0.2$	57.65 $\pm$ 29.48	72.21 $\pm$ 15.97	60.25 $\pm$ 26.86
	$\rho_{m2} = 0.4$	58.00 $\pm$ 29.42	72.46 $\pm$ 15.96	60.57 $\pm$ 26.76
	$\rho_{m3} = 0.6$	57.82 $\pm$ 29.55	72.61 $\pm$ 15.99	60.45 $\pm$ 26.84
	$\rho_{m4} = 0.8$	57.34 $\pm$ 29.72	72.72 $\pm$ 15.95	60.09 $\pm$ 26.97
	$\rho_{m5} = 1.0$	56.90 $\pm$ 29.79	73.01 $\pm$ 15.62	59.78 $\pm$ 26.98
Crossover rate ρ_c (0-1)	$\rho_{c1} = 0.2$	38.77 $\pm$ 32.80	69.58 $\pm$ 15.77	43.26 $\pm$ 30.41
	$\rho_{c2} = 0.4$	54.63 $\pm$ 30.15	72.09 $\pm$ 15.94	57.84 $\pm$ 27.28
	$\rho_{c3} = 0.6$	61.78 $\pm$ 26.75	73.21 $\pm$ 15.89	64.12 $\pm$ 24.10
	$\rho_{c4} = 0.8$	65.22 $\pm$ 24.63	73.83 $\pm$ 15.81	67.07 $\pm$ 22.23
	$\rho_{c5} = 1.0$	67.32 $\pm$ 23.21	74.30 $\pm$ 15.65	68.85 $\pm$ 20.98

Figure 1.6: Box plots, mean values, and standard deviation of the precision, recall, and F-Measure values obtained by each search parameter.

Ranking	Parameters				Performance		
	μ	λ	p_m	p_c	Precision	Recall	F-measure
1	150	50%	1.0	1.0	73.08	75.45	73.64
2	200	50%	1.0	1.0	73.01	75.51	73.62
3	150	60%	1.0	1.0	72.98	75.34	73.53
4	150	40%	1.0	1.0	72.92	75.38	73.53
5	250	60%	1.0	1.0	72.70	75.80	73.53
				...			
621	250	20%	0.2	0.2	7.12	62.53	12.10
622	250	20%	0.4	0.2	6.79	62.24	11.74
623	250	20%	0.6	0.2	6.63	61.91	11.52
624	250	20%	0.8	0.2	6.59	62.33	11.47
625	250	20%	1.0	0.2	6.48	62.09	11.31

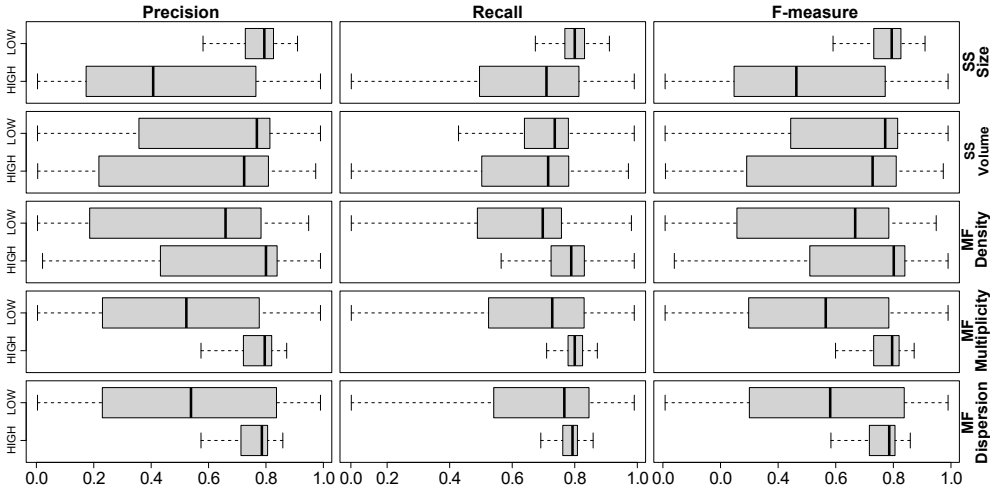
Table 1.3: Search parameter vector performance ranking ordered by F-measure.

Table 1.3 shows the best and worst performance means by each search parameter vector ordered by F-measure. The top five search parameter vectors have the values of mutation rate and the crossover rate in common, (i.e., both are equal to 1.0). The bottom five search parameter vectors have the values of population size ($\mu_5 = 250$), replacement percentage $\lambda_1 = 20\%$, and crossover rate ($p_{c1} = 0.2$) in common. However, the mutation rate takes all possible values.

Performance by MD-MFLP

The bottom part of Figure 1.7 shows the mean values and standard deviation of the precision, recall, and F-measure values obtained by each MD-MFLP. In the measures related to the search space, LOW values obtained better values in performance, while in the measure related to the model fragment, HIGH values obtained better values.

For the SS-Size, the differences between HIGH and LOW values were above 22 points, obtaining a mean of 70.55% in the F-measure for LOW values. For the SS-Volume, the differences between HIGH and LOW values were above 6 points, obtaining a mean of 63.17% in the F-measure for LOW values. For the MF-Density, the differences between HIGH and LOW values were above 14 points, obtaining a mean of 67.41% in the F-measure for HIGH values. For the MF-Multiplicity, the differences between HIGH and LOW values were above 13 points, obtaining a mean of 67.6% in the F-measure for HIGH values. Finally, for the MF-Dispersion, the differences between HIGH and LOW values



MD-MFLP	Value	Precision $\pm$ (σ)	Recall $\pm$ (σ)	F-measure $\pm$ (σ)
SS-Size	LOW	68.79 $\pm$ 24.73	79.16 $\pm$ 5.85	70.55 $\pm$ 21.60
	HIGH	44.46 $\pm$ 29.41	64.97 $\pm$ 19.97	48.22 $\pm$ 27.43
SS-Volume	LOW	60.40 $\pm$ 28.35	74.81 $\pm$ 13.25	63.17 $\pm$ 25.42
	HIGH	54.05 $\pm$ 30.68	69.90 $\pm$ 18.28	56.63 $\pm$ 28.16
MF-Density	LOW	50.59 $\pm$ 30.29	67.50 $\pm$ 17.62	53.08 $\pm$ 28.06
	HIGH	64.53 $\pm$ 27.13	77.74 $\pm$ 11.93	67.41 $\pm$ 23.55
MF-Multiplicity	LOW	50.51 $\pm$ 29.35	66.90 $\pm$ 19.44	53.70 $\pm$ 27.15
	HIGH	65.55 $\pm$ 27.78	79.10 $\pm$ 5.58	67.66 $\pm$ 24.55
MF-Dispersion	LOW	51.93 $\pm$ 30.28	68.77 $\pm$ 19.68	55.22 $\pm$ 27.91
	HIGH	65.03 $\pm$ 26.89	77.72 $\pm$ 5.28	66.91 $\pm$ 23.86

Figure 1.7: Box plots, mean values, and standard deviation of the Precision, Recall, and F-Measure values obtained by each MD-MFLP. Each measure is reported as a two-value factor (HIGH and LOW).

Ranking	Search Space		Model Fragment			Performance		
	Size	Volume	Density	Multiplicity	Dispersion	Precision	Recall	F-measure
1	LOW	LOW	HIGH	LOW	LOW	73.19	84.54	75.55
2	LOW	LOW	HIGH	LOW	HIGH	73.15	79.48	74.35
3	LOW	HIGH	HIGH	LOW	LOW	72.24	82.70	74.26
4	LOW	HIGH	HIGH	HIGH	LOW	71.65	82.68	73.74
5	LOW	HIGH	HIGH	LOW	HIGH	71.83	79.53	73.23
				...				
20	HIGH	LOW	HIGH	LOW	LOW	48.76	70.97	55.28
21	HIGH	LOW	LOW	LOW	HIGH	51.15	71.64	53.34
22	HIGH	HIGH	HIGH	LOW	LOW	49.35	64.73	52.54
23	HIGH	LOW	LOW	LOW	LOW	38.80	54.87	41.47
24	HIGH	HIGH	LOW	LOW	LOW	34.05	55.63	37.40

Table 1.4: MD-MFLP performance ranking ordered by F-measure.

were above 11 points, obtaining a mean of 66.91% in the F-measure for HIGH values.

Table 1.4 shows the ranking of performance means by each MD-MFLP combination ordered by F-measure. Some of the 32 scenarios are missing due to the lack of test cases with those combinations of MD-MFLPs. The top five best results were achieved when the SS-Size was equal to LOW and the MF-Density was equal to HIGH. The bottom five results coincide in SS-Size with the opposite value (equal to HIGH) and the MF-Multiplicity value equal to LOW.

Performance by search parameter value and MD-MFLP

Table 1.5 presents the mean values of precision, recall, and F-measure achieved by each search parameter value and each MD-MFLP. Similarly, Figure 1.8 shows the graphs obtained from those values. The results show that the performance of MD-MFLP remained the same as in the previous subsection. In the measures related to the search space, LOW values obtained better values in performance, while, in the measure related to the model fragment HIGH values obtained better values.

The same occurred with the population size, replacement percentage, and crossover rate search parameters. The values that obtained the best performance for them are the ones described in section 1.4.1. However, the best performing values for mutation rate were not those described in section 1.4.1 for all cases. The $p_{m1} = 0.2$ achieved the best results in terms of precision for SS-Size, MF-Multiplicity, and MF-Dispersion.

		Population size (μ)					Replacement percentage (λ)					Mutation rate (p_m)					Crossover rate (p_c)				
		50	100	150	200	250	20%	30%	40%	50%	60%	0.2	0.4	0.6	0.8	1.0	0.2	0.4	0.6	0.8	1.0
		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
SS-Size	LOW	79.54	77.64	72.49	62.96	51.32	49.04	65.96	74.01	76.87	78.06	69.94	69.66	69.06	68.21	67.08	53.05	67.29	72.45	74.86	76.29
		79.85	79.93	79.65	78.85	77.53	76.84	79.04	79.80	80.02	80.12	78.95	79.12	79.21	79.27	79.28	77.19	78.99	79.62	79.91	80.10
	F	79.65	78.10	73.83	65.72	55.46	53.10	68.32	75.19	77.58	78.57	71.48	71.24	70.76	70.08	69.21	56.86	69.34	73.74	75.79	77.02
HIGH	P	55.56	51.36	45.16	39.03	31.20	28.02	40.39	48.09	52.01	53.80	43.36	44.44	44.75	44.71	45.05	22.16	39.90	49.36	54.00	56.88
	R	65.60	65.44	65.23	64.73	63.84	62.11	64.50	65.70	66.18	66.36	64.38	64.73	64.93	65.09	65.72	60.73	64.06	65.75	66.76	67.56
	F	58.06	54.38	48.99	43.40	36.25	32.96	44.64	51.67	55.13	56.70	47.19	48.16	48.47	48.47	48.81	27.44	44.47	52.92	56.92	59.34
SS-Volume	P	71.93	69.00	63.01	54.33	43.72	41.61	56.72	65.00	68.52	70.14	60.73	60.99	60.70	60.12	59.46	42.10	57.43	64.43	67.93	70.10
	R	75.67	75.61	75.23	74.39	73.13	72.16	74.49	75.50	75.86	76.03	74.48	74.73	74.83	74.91	75.09	72.02	74.35	75.38	75.95	76.33
	F	73.08	70.62	65.56	58.01	48.59	46.33	60.11	67.35	70.35	71.71	63.41	63.64	63.40	62.95	62.45	46.85	60.79	66.84	69.78	71.59
HIGH	P	64.19	61.18	56.00	48.92	39.93	36.52	50.99	58.39	61.52	62.81	53.88	54.34	54.30	53.95	53.76	34.69	51.19	58.53	61.91	63.91
	R	70.31	70.32	70.23	69.81	68.84	67.43	69.66	70.56	70.89	70.97	69.44	69.70	69.89	70.03	70.46	66.59	69.32	70.54	71.25	71.82
	F	65.50	62.87	58.43	52.21	44.12	40.68	54.02	60.61	63.35	64.48	56.38	56.82	56.84	56.58	56.50	38.87	54.23	60.79	63.75	65.49
MF-Density	P	62.24	58.85	52.60	44.46	34.77	32.38	46.95	55.00	58.53	60.08	50.62	50.99	50.82	50.35	50.15	30.65	47.72	55.28	58.62	60.66
	R	68.11	68.08	67.83	67.23	66.23	65.12	67.25	68.13	68.43	68.56	67.11	67.35	67.46	67.56	68.00	64.57	67.04	68.10	68.64	69.12
	F	63.53	60.49	55.01	47.66	38.74	36.08	49.89	57.26	60.40	61.79	53.02	53.40	53.29	52.92	52.79	34.39	50.64	57.53	60.52	62.34
MF-Multiplicity	P	74.69	72.16	67.15	59.37	49.30	46.30	61.38	69.09	72.26	73.65	64.72	65.05	64.86	64.37	63.68	46.94	61.57	68.30	71.85	74.01
	R	78.44	78.42	78.17	77.46	76.20	74.97	77.41	78.46	78.85	78.99	77.34	77.61	77.79	77.90	78.05	74.62	77.16	78.34	79.05	79.52
	F	75.84	73.82	69.74	63.19	54.46	51.54	64.90	71.41	74.04	75.16	67.51	67.79	67.66	67.29	66.79	52.18	65.08	70.74	73.65	75.39
MF-Multiplicity	P	58.90	56.01	51.94	46.47	39.23	34.92	47.76	54.36	57.14	58.37	49.68	50.56	50.76	50.65	50.89	31.14	47.18	54.83	58.57	60.82
	R	67.22	67.17	67.13	66.84	66.12	64.58	66.61	67.51	67.83	67.95	66.37	66.67	66.86	67.02	67.56	63.39	66.17	67.53	68.37	69.02
	F	61.02	58.50	55.01	50.24	43.76	39.60	51.39	57.22	59.63	60.69	52.91	53.71	53.92	53.87	54.12	35.92	51.01	57.78	60.97	62.84
MF-Distribution	P	79.33	76.28	68.88	58.08	45.19	44.34	61.41	70.76	74.75	76.49	66.72	66.48	65.86	64.96	63.73	47.47	63.10	69.68	72.79	74.72
	R	80.14	80.14	79.65	78.59	76.99	76.25	78.82	79.85	80.22	80.37	78.87	79.06	79.16	79.21	79.22	76.63	78.83	79.68	80.05	80.32
	F	79.52	76.97	70.71	61.29	49.79	48.56	64.18	72.41	75.83	77.31	68.60	68.40	67.90	67.17	66.22	51.62	65.62	71.34	74.02	75.69
MF-Distribution	P	61.34	58.24	53.50	47.27	39.28	35.36	48.82	55.96	59.07	60.44	51.23	52.04	52.20	52.03	52.14	31.97	48.48	56.37	60.25	62.57
	R	69.26	69.20	69.07	68.61	67.71	66.15	68.43	69.45	69.84	69.97	68.24	68.55	68.75	68.89	69.41	65.08	68.04	69.45	70.31	70.96
	F	63.36	60.69	56.66	51.24	44.13	40.27	52.61	58.89	61.57	62.74	54.53	55.27	55.45	55.34	55.50	37.00	52.43	59.37	62.67	64.61
HIGH	P	77.93	75.15	68.33	58.07	45.66	44.62	61.24	70.12	73.78	75.39	66.22	65.94	65.32	64.43	63.24	47.84	62.82	68.99	71.85	73.64
	R	78.61	78.61	78.21	77.30	75.86	75.21	77.50	78.39	78.68	78.81	77.51	77.68	77.77	77.81	77.81	75.57	77.49	78.22	78.54	78.76
	F	78.08	75.74	69.94	60.95	49.85	48.48	63.73	71.56	74.71	76.08	67.88	67.65	67.13	66.42	65.48	51.62	65.07	70.44	72.93	74.50

Table 1.5: Mean values for the Precision (P), Recall (R), and F-measure (F) by each search parameter value and each MD-MFLP.

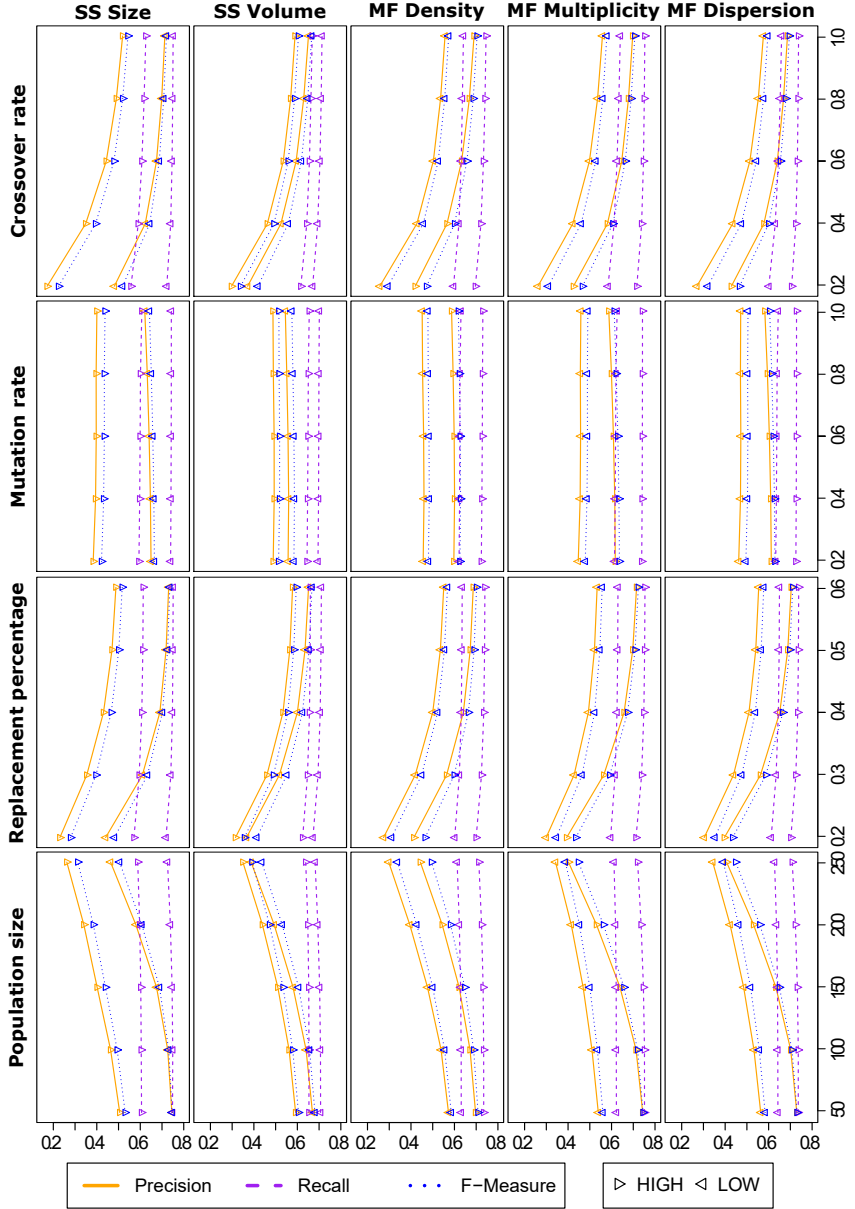


Figure 1.8: Graphs of the Precision, Recall, and F-Measure values obtained by each MD-MFLP (HIGH and LOW) and each search parameter.

Ranking	Search Space		Model Fragment			Parameters				Performance		
	Size	Volume	Density	Multiplicity	Dispersion	λ	μ	p_m	p_c	P	R	F
1	LOW	LOW	HIGH	LOW	LOW	50	50%	1.0	1.0	86.60	86.60	86.60
2	LOW	LOW	HIGH	LOW	LOW	100	30%	1.0	1.0	86.56	86.56	86.56
3	LOW	LOW	HIGH	LOW	LOW	200	40%	0.6	1.0	86.55	86.55	86.55
4	LOW	LOW	HIGH	LOW	LOW	200	60%	1.0	1.0	86.55	86.55	86.55
5	LOW	LOW	HIGH	LOW	LOW	150	40%	0.4	1.0	86.51	86.51	86.51
					...							
14996	HIGH	LOW	LOW	LOW	LOW	200	20%	0.2	0.2	2.30	48.63	4.37
14997	HIGH	LOW	LOW	LOW	LOW	250	20%	0.4	0.2	2.25	51.51	4.29
14998	HIGH	LOW	LOW	LOW	LOW	250	20%	0.2	0.2	2.23	51.21	4.26
14999	HIGH	LOW	LOW	LOW	LOW	250	20%	0.8	0.2	2.21	50.01	4.21
15000	HIGH	LOW	LOW	LOW	LOW	250	20%	0.6	0.2	2.21	50.30	4.21

Table 1.6: Search parameter vector and MD-MFLP performance ranking ordered by F-measure.

Table 1.6 shows the best and worst performance means for each combination of the search parameter vector and MD-MFLP ordered by the F-measure value. The top five values of performance correspond to the same scenario. Again, the scenario, shares that the SS-Size value was LOW and the MF-Density value was HIGH. With regard to the search parameter values, the top five values of performance had the same crossover rate value ($p_{c5} = 1.0$). Similarly, the bottom five values correspond to the same scenario. In contrast to the previous scenario, the SS-Size value was HIGH and the MF-Density value was LOW. Moreover, the bottom five values had the same crossover rate value ($p_{c1} = 0.2$), the same population size value ($\mu_5 = 250$), and the same replacement percentage value ($\lambda_1 = 20\%$). Note that the crossover value was the complete opposite of the previous one ($p_{c1} = 0.2$ bottom values vs. $p_{c5} = 1.0$ top values).

1.4.2 Statistical analysis

The data resulting from the empirical analysis has been analyzed using statistical methods following the guidelines in [2]. The goals of the analysis are: (1) to provide formal and quantitative evidence (statistical significance) regarding whether or not the different search parameter vectors and the different metrics have an impact on the performance; and (2) to show that those differences are significant in practice (effect size).

To enable statistical analysis, the algorithm must be run a large enough number of times (in an independent way) to collect information on the probability distribution for each search parameter vector. Then, a statistical test is run to assess whether there is enough empirical evidence to claim (with a high level of confidence) that there is a difference in the performance.

Statistical significance

To analyze the statistical significance of the different search parameter vectors and the different measures, we used the ANOVA test. To apply this test, it is important to comply with normality and homogeneity of variance. The box plots depicted in the previous sections show that the data does not follow a normal distribution. However, the Central Limit Theorem states that the sampling distribution of means is normally distributed for large enough samples [34]. In our case, the normality of sampling distributions is ensured by having sufficiently large sample sizes.

The result of the ANOVA test showed that there were statistically significant differences between groups because, $p - values$ obtained by precision, recall, and F-measure values were lower than 0.05 for all of the search parameter values and MD-MFLP.

As we detected significant differences between the groups, we tested which groups were significantly different. We performed an additional post hoc analysis, which consisted of a pair-wise comparison among the results for each search parameter and for each MD-MFLP in order to determine the statistically significant differences among the results of each pair of search parameter values and of each MD-MFLP. To do that, we used the Tukey HSD (Honestly Significant Difference) [35] post hoc test, which is designed to test all of the means against each other pairwise. In other words, each mean is compared, in turn, with every other mean. Table 1.7 shows the $p - values$ of the Tukey HSD test comparing the different values for each search parameter and for each MD-MFLP. In addition, the *diff* value shows the difference in the means of the two groups.

The largest differences for the population size parameter were between the values of $\mu_5 = 250$ and $\mu_1 = 50$. For the replacement parameter, they were between the values of $\lambda_5 = 60\%$ and $\lambda_1 = 20\%$. For the mutation rate parameter, they were between the values of $p_{m2} = 0.4$ and $p_{m5} = 1.0$, and, for the crossover rate parameter, they were between the values of $p_{c1} = 0.2$ and $p_{c5} = 1.0$. With regard to the MD-MFLP, the SS-Size showed the biggest differences between HIGH and LOW values.

		Tukey HSD						
		Precision		Recall		F-measure		
		<i>diff</i>	<i>p</i>	<i>diff</i>	<i>p</i>	<i>diff</i>	<i>p</i>	
Search parameters	Population size (μ)	100 - 50	2.96	0.00	0.03	0.97	2.53	0.00
		150 - 50	8.60	0.00	0.28	0.00	7.32	0.00
		200 - 50	16.56	0.00	0.93	0.00	14.27	0.00
		250 - 50	26.44	0.00	2.06	0.00	23.09	0.00
		150 - 100	5.63	0.00	0.25	0.00	4.78	0.00
		200 - 100	13.59	0.00	0.90	0.00	11.73	0.00
		250 - 100	23.47	0.00	2.03	0.00	20.56	0.00
		200 - 150	7.96	0.00	0.65	0.00	6.95	0.00
		250 - 150	17.84	0.00	1.78	0.00	15.77	0.00
		250 - 200	9.88	0.00	1.13	0.00	8.82	0.00
	Replacement percentage (λ)	0.3 - 0.2	14.82	0.00	2.29	0.00	13.58	0.00
		0.4 - 0.2	22.70	0.00	3.25	0.00	20.53	0.00
		0.5 - 0.2	26.05	0.00	3.59	0.00	23.41	0.00
		0.6 - 0.2	27.52	0.00	3.72	0.00	24.67	0.00
		0.4 - 0.3	7.88	0.00	0.96	0.00	6.95	0.00
		0.5 - 0.3	11.23	0.00	1.31	0.00	9.83	0.00
		0.6 - 0.3	12.70	0.00	1.44	0.00	11.08	0.00
		0.5 - 0.4	3.35	0.00	0.34	0.00	2.88	0.00
		0.6 - 0.4	4.82	0.00	0.48	0.00	4.14	0.00
	0.6 - 0.5	1.47	0.00	0.13	0.03	1.26	0.00	
	Mutation rate (p_m)	0.4 - 0.2	0.35	0.00	0.25	0.00	0.33	0.00
		0.6 - 0.2	0.17	0.26	0.40	0.00	0.21	0.06
		0.8 - 0.2	0.31	0.00	0.50	0.00	0.16	0.23
		1.0 - 0.2	0.75	0.00	0.80	0.00	0.47	0.00
		0.6 - 0.4	0.18	0.24	0.15	0.01	0.12	0.54
		0.8 - 0.4	0.66	0.00	0.25	0.00	0.49	0.00
		1.0 - 0.4	1.10	0.00	0.54	0.00	0.80	0.00
		0.8 - 0.6	0.48	0.00	0.10	0.15	0.37	0.00
		1.0 - 0.6	0.93	0.00	0.40	0.00	0.68	0.00
	1.0 - 0.8	0.45	0.00	0.29	0.00	0.31	0.00	
	Crossover rate (p_c)	0.4 - 0.2	15.85	0.00	2.51	0.00	14.58	0.00
		0.6 - 0.2	23.00	0.00	3.63	0.00	20.86	0.00
		0.8 - 0.2	26.45	0.00	4.25	0.00	23.81	0.00
		1.0 - 0.2	28.55	0.00	4.73	0.00	25.59	0.00
		0.6 - 0.4	7.15	0.00	1.12	0.00	6.28	0.00
		0.8 - 0.4	10.59	0.00	1.75	0.00	9.22	0.00
		1.0 - 0.4	12.69	0.00	2.22	0.00	11.01	0.00
		0.8 - 0.6	3.44	0.00	0.63	0.00	2.95	0.00
		1.0 - 0.6	5.54	0.00	1.10	0.00	4.73	0.00
	1.0 - 0.8	2.10	0.00	0.47	0.00	1.78	0.00	
MD-MFLP	SS-Size	LOW - HIGH	24.33	0.00	14.19	0.00	22.33	0.00
	SS-Volume	LOW - HIGH	6.35	0.00	4.90	0.00	6.54	0.00
	MF-Density	LOW - HIGH	13.95	0.00	10.24	0.00	14.32	0.00
	MF-Multiplicity	LOW - HIGH	15.04	0.00	12.21	0.00	13.95	0.00
	MF-Dispersion	LOW - HIGH	13.10	0.00	8.95	0.00	11.70	0.00

Table 1.7: Tukey HSD test values for each search parameter and for each MD-MFLP

		Cohen d value		
		Precision	Recall	F-Measure
Search parameters	Population size (μ)	0.964	0.130	0.916
	Replacement percentage (λ)	0.971	0.235	0.948
	Mutation rate (p_m)	0.037	0.034	0.030
	Crossover rate (p_c)	1.005	0.301	0.980
MD-MFLP	SS-Size	0.901	0.997	0.913
	SS-Volume	0.216	0.312	0.245
	MF-Density	0.485	0.680	0.553
	MF-Multiplicity	0.525	0.831	0.537
	MF-Dispersion	0.454	0.586	0.445

Table 1.8: Cohen d values for each search parameter and for each MD-MFLP.

Effect size

Apart from determining if two search parameter values and two MD-MFLPs are significantly different in performance, it is also important to assess the magnitude of those differences. To do so, we used effect size measures; specifically, we applied the Cohen d value [6].

Table 1.8 contains the Cohen d values for each search parameter and for each MD-MFLP. Cohen d values above 0.8 mean high interaction effect (colored gray), values from 0.5 to 0.8 mean medium interaction effect, and values from 0.2 to 0.5 mean low interaction effect. Values from 0 to 0.2 mean almost no interaction effect.

The Cohen d values of population size, replacement percentage, and crossover rate parameters affirm that those search parameters have a high interaction effect in precision and F-measure but not in recall. However, the mutation rate parameter did not have interaction in any of the performance measures. In contrast, all MD-MFLPs had a higher effect in recall than in precision. The SS-Size measure had a high effect in precision, recall, and F-measure.

1.4.3 Findings and discussion

The evaluation presented in this paper provides a detailed analysis of the performance obtained by the results of the study of different search parameters of a search algorithm to locate 1,895 MFLPs extracted from two different industrial domains. The results obtained in the evaluation allow us to answer each of the research questions.

RQ1. Is there any impact on performance regarding the search parameter vector for model fragment location problems?

The population size, replacement percentage, and crossover rate parameters have large effects on the precision value while the mutation rate parameter has a small effect. The population size parameter and the precision value are inversely proportional. In other words, the precision value increases as the population size parameter decreases. The replacement percentage parameter and the crossover rate parameter are directly proportional to the precision value, i.e., the precision increases as they increase. However, all search parameters had small effects on the recall value.

The results show that the best mean performance was obtained by population size $\mu_3 = 150$, replacement percentage $\lambda_4 = 50\%$, mutation rate $p_{m5} = 1.0$, and crossover rate $p_{c5} = 1.0$. If we consider the search parameters separately, the population size value, the replacement percentage, and the mutation rate vary to achieve the best performance. However, the crossover value remained at $p_{c5} = 1.0$.

Our statistical analysis of the results confirms that the crossover rate parameter is the one that presents larger differences and higher effect size. The differences observed in the results when switching the value for the crossover rate are significant and yield large differences in the performance. Higher values of crossover rate yield to higher values of performance (around a 25% improvement in F-measure).

We analyzed the results to understand why the influence of the crossover rate on the performance of the algorithm is significant and the influence of the mutation rate is negligible and not statistically significant. It turns out that crossover is more beneficial than mutation due to the characteristics of the model fragments. Our research shows that the model fragments involve elements that are not always connected to each other. In other words, the location of a relevant model fragment requires identifying clusters of elements distributed throughout the model. For instance, the functionality related to braking in the CAF trains is not restricted to a specific part of the model

and, therefore, affects numerous elements throughout. While the mutation operation has the possibility of incorporating elements from any part of the model, the crossover operation significantly favors gathering the relevant model elements. The utilization of real-world industry models has enabled the identification of this finding. It may be challenging to identify it in academic instances as they are simpler and the model elements are interconnected. This is partly because academic research models and model fragments are smaller in size compared to industrial models.

RQ2. Is there any impact on performance regarding the measures that define the model fragment location problems?

There were differences in the performance metrics between the HIGH and LOW values of the MD-MFLP. The SS-Size measure obtained the highest differences for precision and recall. The MF-Multiplicity had a large effect on the recall value but a medium effect on the precision value. MF-Density and MF-Multiplicity had medium effects on precision and recall values. The SS-Volume measure obtained the smallest differences for precision and recall.

The measures related to the search space (SS-Size and SS-Volume) obtained the best values in performance with the LOW values. However, the measures that are related to the solution (MF-Density, MF-Multiplicity, and MF-Dispersion) obtained the best values in performance with the HIGH values. The localization problems used in the evaluation have been classified using the five MD-MFLPs measures analyzed. This information can be very useful for other software engineers facing search problems in order to decide which parameters to use in their evolutionary algorithms.

RQ3. Are there any relations between the measures that define the model fragment location problems and the search parameter vector used that affect the performance?

There are interactions between the search parameter vector and the MD-MFLP that affect the performance metrics. The highest interaction between the MD-MFLP and the search parameter vector is produced by different values of SS-Size. The crossover rate is the most important parameter value, which maintains its value in many scenarios.

In addition, this evaluation allowed us to realize that the nature of the search problems has a greater effect on recall, while the search strategy has a greater effect on the precision of the solutions found.

Regarding generalization, we have performed an evaluation of 1,895 localization problems from two different industrial domains classified using the five MD-MFLPs measures. Although the general trends are maintained, the parameter vector that offers better results depends on each scenario. The classification of localization problems can serve as a practical guide, enabling more informed decisions in evolutionary algorithm design from other model domains.

1.4.4 Threats to validity

This section presents some threats to the validity of the results presented. We have followed the guidelines suggested by De Oliveira et. al [25].

Conclusion validity: We identified three threats of this type. The first threat is *not accounting for random variation*. To address this threat, we considered 100 independent runs for each of the test cases for each of the search parameter vectors. The second threat is the *lack of formal hypotheses and statistical tests*. In this work, we employed standard statistical analysis following accepted guidelines [2] to avoid this threat. The third threat is the *lack of good descriptive statistics*. In this work, we have used precision, recall, and F-measure metrics to analyze the confusion matrix obtained from the experiments; however, other metrics could be applied.

Internal validity: We identified four threats of this type. The first identified threat of this type is the *poor search parameter settings*. In this work, we evaluated which search parameter values worked best when performing model fragment location with our algorithm. In addition, the choice of the k value in the application of SVD can produce sub-optimal accuracy when using LSI for software artifacts [26]. The second threat is the *lack of real problem instances*. The evaluation of this paper was applied to 1,895 location problems from two industrial case studies, BSH and CAF. The third threat is the *lack of clear data collection tools and procedures*. The set of 1,895 feature localization problems used in the evaluation has been provided by our industrial partners BSH and CAF. The test cases provided are representative of their respective domains: induction hobs and railway domains. The fourth threat is the *lack of discussion on code instrumentation*. The evolutionary algorithm used in our evaluation is not the contribution of this paper. It was presented in [12] and improved in [16] where the source code of the algorithm was made public.

Construct validity: We identified one threat of this type. The identified threat is the *lack of assessing the validity of cost measures*. To address this threat, we performed a fair comparison among the algorithms with different search parameter values by generating the same number of model fragments and allocating the same budget time. Furthermore, the precision, recall, and F-measure measures used for solution quality are widely used in the information retrieval field [31].

External validity: We identified three threats of this type. The first is the *lack of a clear object selection strategy* and the second is the *lack of evaluations for instances of growing size and complexity*. To mitigate these threats, we used a large number of case studies from two industrial partners (BSH and CAF). Our instances are extracted from real-world problems. Also, the approach was evaluated in two different domains that varied in size and complexity. The third threat is the *lack of a clear definition of target instances*. We are concerned with the generalization of our findings, hence, we classify our location problems using the measures that define model fragment location problems (MD-MFLP) presented in [4]. However, the generalization could be affected since we only address location problems in models neglecting other kinds of artifacts.

1.5 Related work

Some works have investigated the effect of the use of different operators that are tailored to the domain on the results of the search being performed [7, 17, 29, 28]. For instance, [29] extends their previous work by defining software architectures [30] to analyze the effect of the crossover operator in genetic algorithms that are used to synthesize software architecture designs. The authors compare sexual and asexual crossover operators, applying them to two test cases to conclude that the asexual crossover (i.e., no crossover) provides better results for that domain than a regular sexual crossover. However, the work was further refined [28] to propose a complementary crossover that is capable of yielding solutions with better quality than the asexual crossover. The crossover operator proposed can find complementary parents and produce offspring that combine the best from both parents.

Similarly, [7] introduces a feature-driven crossover operator that is capable of providing better results when applied to optimize a product line architecture. The study compares a multi-objective evolutionary algorithm that is applied

to two case studies using their crossover operator or no crossover operator at all.

Harman et al. [17] propose a new crossover operator to be applied in the context of automated software re-modularization. The new crossover operator aims to preserve building blocks from parents that are transferred without modifications to the new offspring. This leads to better results than a regular crossover operator.

These works can improve the results by using a different operator rather than by changing the values of the parameters of the operator. The different operators that are applied by the evolutionary algorithm in those works correspond to qualitative parameters. In contrast, the focus in our work are the quantitative parameters of the evolutionary algorithm, looking for the values that provide the best results with current genetic operators, which leads to great improvement.

Some works focus on the tuning of the parameters of evolutionary algorithms [11, 3, 10]. For instance, [11] presents a survey on parameter tuning and parameter control and discusses several techniques to achieve it. The authors also make the distinction between parameter tuning (how to choose parameters before running the search algorithm) and parameter control (how to change parameters while the search is being performed). However, it is important to note that parameter control does not fully address the problems of parameter tuning as the introduction of the control mechanisms usually leads to more parameters that need to be set.

The work from [3] deals directly with the *No Free Lunch theorem* [36] (it is impossible to tune a search algorithm so that it will have optimal parameter values for all problem instances). The authors perform a large empirical analysis in the context of test data generation for object-oriented software to determine the impact of the tuned parameters on the searches. They conclude that parameter tuning affects the performance of search algorithms; however, well-tuned parameters are complex to find, and default values may be enough.

In [10], the authors carry out a more general study of the parameter tuning of evolutionary algorithms. Contrary to [3], they conclude that by using tuning algorithms one cannot only obtain superior parameter values, but also a lot of information about problem instances, parameter values, and algorithm performance.

Nevertheless, those works are applied to the (more general) field of Search-Based Software Engineering and therefore do not take the particularities of

the SBMDE field into account. It is not clear whether search parameter values that provide good results in problem instances that do not deal with models will behave similarly with SBMDE problem instances.

1.6 Conclusion

More and more, researchers are reformulating MDE activities as search problems. These works use Search-Based optimization techniques (mainly those from the evolutionary computation literature) to automate the search for optimal and near-optimal solutions. However, these works are neglecting the influence of the search parameter values selected and the nature of their problems in their results.

In this work, we have performed an evaluation to determine the impact of different search parameter values when performing model fragment location following an evolutionary algorithm and taking into account the nature of the problems to locate. It turns out that there are interactions between the search parameter vector and the MD-MFLP that affect the performance metrics. Different values of population size, replacement percentage, or crossover rate parameters produce variations of around 30% in performance, but the mutation rate parameter produces differences of less than 5% in performance. In addition, LOW values related to the search space and HIGH values related to the model fragment in the MD-MFLPs obtain better values in performance. To achieve these results, the approach has been tested using 625 different search parameter vectors applied to 1,895 different MFLPs. In addition, the results have been supported with a statistical analysis that determines that the results are significant and are not due to mere chance.

Finally, we think that the SBMDE community could benefit from the result of our work by taking into account the influence of search parameter values and the nature of the MFLPs in their results.

Bibliography

- [1] Lorena Arcega et al. “An approach for bug localization in models using two levels: model and metamodel”. In: *Softw. Syst. Model.* 18.6 (2019), pp. 3551–3576. DOI: 10.1007/s10270-019-00727-y (cit. on p. 24).

- [2] Andrea Arcuri and Lionel Briand. “A Hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering”. In: *Software Testing, Verification and Reliability* 24.3 (2014), pp. 219–250. DOI: <https://doi.org/10.1002/stvr.1486>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1486> (cit. on pp. 48, 54).
- [3] Andrea Arcuri and Gordon Fraser. “Parameter tuning or default values? An empirical investigation in search-based software engineering”. In: *Empirical Software Engineering* 18.3 (2013), pp. 594–623. ISSN: 1573-7616. DOI: 10.1007/s10664-013-9249-9 (cit. on pp. 32, 33, 56).
- [4] Manuel Ballarín et al. “Measures to Report the Location Problem of Model Fragment Location”. In: *21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. MODELS ’18*. Copenhagen, Denmark: ACM, 2018, pp. 189–199. ISBN: 978-1-4503-4949-9 (cit. on pp. 25, 30, 55).
- [5] Ilhem Boussaïd, Patrick Siarry, and Mohamed Ahmed-Nacer. “A survey on search-based model-driven engineering”. In: *Automated Software Engineering* 24.2 (2017), pp. 233–294. ISSN: 1573-7535. DOI: 10.1007/s10515-017-0215-4 (cit. on p. 24).
- [6] Jacob Cohen. “Statistical power analysis”. In: *Current directions in psychological science* 1.3 (1992), pp. 98–101 (cit. on p. 51).
- [7] Thelma Elita Colanzi and Silvia Regina Vergilio. “A feature-driven crossover operator for multi-objective and evolutionary optimization of product line architectures”. In: *Journal of Systems and Software* 121 (2016), pp. 126–143. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2016.02.026> (cit. on p. 55).
- [8] Jorge Echeverría et al. “Comparison of Search Strategies for Feature Location in Software Models”. In: *J. Syst. Softw.* 181.C (2021). ISSN: 0164-1212. DOI: 10.1016/j.jss.2021.111037 (cit. on pp. 24, 30).
- [9] A. E. Eiben and S. K. Smit. “Evolutionary Algorithm Parameters and Methods to Tune Them”. In: *Autonomous Search*. Ed. by Youssef Hamadi, Eric Monfroy, and Frédéric Saubion. Berlin, Heidelberg: Springer Berlin

- Heidelberg, 2012, pp. 15–36. ISBN: 978-3-642-21434-9. DOI: 10.1007/978-3-642-21434-9_2 (cit. on p. 31).
- [10] A.E. Eiben and S.K. Smit. “Parameter tuning for configuring and analyzing evolutionary algorithms”. In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 19–31. ISSN: 2210-6502. DOI: <https://doi.org/10.1016/j.swevo.2011.02.001> (cit. on p. 56).
- [11] Agoston E Eiben et al. “Parameter control in evolutionary algorithms”. In: *Parameter setting in evolutionary algorithms*. Springer, 2007, pp. 19–46 (cit. on p. 56).
- [12] Jaime Font et al. “Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering”. In: *IEEE Transactions on Evolutionary Computation* PP.99 (2017), pp. 1–1. ISSN: 1089-778X. DOI: 10.1109/TEVC.2017.2751100 (cit. on pp. 24, 34, 40, 54).
- [13] Jaime Font et al. “Building software product lines from conceptualized model patterns”. In: *Proceedings of the 19th International Conference on Software Product Line (SPLC)*. 2015, pp. 46–55. DOI: 10.1145/2791060.2791085 (cit. on p. 24).
- [14] Jaime Font et al. “Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm”. In: *Software Reuse: Bridging with Social-Awareness - 15th International Conference, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, Proceedings*. 2016, pp. 39–54. DOI: 10.1007/978-3-319-35122-3_3 (cit. on p. 32).
- [15] Jaime Font et al. “Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques”. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. MODELS ’16. Saint-malo, France: ACM, 2016, pp. 272–282. ISBN: 978-1-4503-4321-3. DOI: 10.1145/2976767.2976789 (cit. on pp. 29, 32, 40).
- [16] Jaime Font et al. “Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility”. In: *Software and Systems Modeling* (2021). ISSN: 1619-1374. DOI: 10.1007/s10270-021-00870-5 (cit. on pp. 24, 29, 54).

- [17] Mark Harman, Robert M. Hierons, and Mark Proctor. “A New Representation And Crossover Operator For Search-based Optimization Of Software Modularization”. In: *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*. 2002, pp. 1351–1358 (cit. on pp. 55, 56).
- [18] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Information and software Technology* 43.14 (2001), pp. 833–839 (cit. on p. 24).
- [19] Stuart Kent. “Model Driven Engineering”. In: *Integrated Formal Methods*. Ed. by Michael Butler, Luigia Petre, and Kaisa Sere. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 286–298. ISBN: 978-3-540-47884-3 (cit. on p. 24).
- [20] Thomas K Landauer, Peter W Foltz, and Darrell Laham. “An introduction to latent semantic analysis”. In: *Discourse processes* 25.2-3 (1998), pp. 259–284 (cit. on p. 29).
- [21] Robert C MacCallum et al. “On the practice of dichotomization of quantitative variables.” In: *Psychological methods* 7.1 (2002), p. 19 (cit. on p. 38).
- [22] Christopher Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999 (cit. on p. 29).
- [23] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Cambridge university press Cambridge, 2008 (cit. on p. 25).
- [24] Jaechang Nam and Sunghun Kim. “CLAMI: Defect Prediction on Unlabeled Datasets (T)”. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015, pp. 452–463. DOI: 10.1109/ASE.2015.56 (cit. on p. 38).
- [25] Márcio de Oliveira Barros and Arilo Cláudio Dias-Neto. “0006/2011-Threats to Validity in Search-based Software Engineering Empirical Studies”. In: *RelaTe-DIA* 5.1 (2011) (cit. on p. 54).

- [26] A. Panichella et al. “Parameterizing and Assembling IR-Based Solutions for SE Tasks Using Genetic Algorithms”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. 2016, pp. 314–325. DOI: 10.1109/SANER.2016.97 (cit. on p. 54).
- [27] Francisca Pérez et al. “Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study”. In: *Inf. Softw. Technol.* 103 (2018), pp. 188–201 (cit. on p. 24).
- [28] O. Räihä, K. Koskimies, and E. Mäkinen. “Complementary crossover for genetic software architecture synthesis”. In: *2010 10th International Conference on Intelligent Systems Design and Applications*. 2010, pp. 266–271. DOI: 10.1109/ISDA.2010.5687255 (cit. on p. 55).
- [29] Outi Räihä, Kai Koskimies, and Erkki Mäkinen. “Empirical study on the effect of crossover in genetic software architecture synthesis”. In: *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*. 2009, pp. 619–625. DOI: 10.1109/NABIC.2009.5393431 (cit. on p. 55).
- [30] Outi Räihä et al. “Pattern-based Genetic Model Refinements in MDA”. In: *Nordic J. of Computing* 14.4 (Dec. 2008), pp. 338–355. ISSN: 1236-6064 (cit. on p. 55).
- [31] Gerard Salton and Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. In: *Information processing & management* 24.5 (1988), pp. 513–523 (cit. on p. 55).
- [32] Abdel Salam Sayyad et al. “On Parameter Tuning in Search Based Software Engineering: A Replicated Empirical Study”. In: *Proceedings of the 2013 3rd International Workshop on Replication in Empirical Software Engineering Research*. RESER ’13. USA: IEEE Computer Society, 2013, 84–90. ISBN: 9780769551210. DOI: 10.1109/RESER.2013.6 (cit. on pp. 32, 33).
- [33] Stephen V. Stehman. “Selecting and interpreting measures of thematic classification accuracy”. In: *Remote Sensing of Environment* 62.1 (1997), pp. 77–89. ISSN: 0034-4257. DOI: 10.1016/S0034-4257(97)00083-7 (cit. on p. 38).

- [34] Barbara G Tabachnick and Linda S Fidell. *Experimental designs using ANOVA*. Vol. 724. Thomson/Brooks/Cole Belmont, CA, 2007 (cit. on p. 49).
- [35] John W. Tukey. “Comparing individual means in the analysis of variance.” In: *Biometrics* 5 2 (1949), pp. 99–114 (cit. on p. 49).
- [36] D. H. Wolpert and W. G. Macready. “No free lunch theorems for optimization”. In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. ISSN: 1089-778X. DOI: 10.1109/4235.585893 (cit. on p. 56).

Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games

The development of video games usually involves two main methods: Code-centric Development (CcD) and Model-Driven Development (MDD). CcD uses code languages that provide more control but it requires more effort in order to deal with implementation details. MDD raises the abstraction level (avoiding implementation details) by means of software models that are transformed into code or interpreted at run-time. Raising the abstraction level favors the participation of non-technical roles such as level designers or artists that are essential for video game development. However, bug localization, which is crucial for identifying faults, is less explored in MDD despite its advantages. This work examines how MDD and CcD impact bug localization by using a commercial video game. We compare bug localization in terms of performance, productivity, and user satisfaction. The results showed that bug localization in MDD led to higher satisfaction among subjects. However, the differences in performance or productivity depended on experience and favored CcD for professionals. Our findings suggest that bug localization practices performed suboptimally in models, indicating a knowledge gap in addressing bugs within MDD environments. With the rising popularity of MDD in video games, there is a need to explore alternative forms of bug localization for MDD.

Author version of the article: I. Roca, A. Domingo, O. Pastor, C. Cetina, and L. Arcega “*Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games*”. In: Guizzardi, G., Santoro, F., Mouratidis, H., Soffer, P. (eds) *Advanced Information Systems Engineering. CAiSE 2024. Lecture Notes in Computer Science*, Volume 14663. Springer, Cham. https://doi.org/10.1007/978-3-031-61057-8_34.

2.1 Introduction

The video game industry has evolved into a highly profitable segment¹. The increasing popularity of video games has led to a rise in the complexity of creating them. Video game development involves complex software that encompasses various aspects of gaming, such as game mechanics, graphics, and physics. As a result, a separate field of engineering, known as Game Software Engineering (GSE), emerged. Its focus lies in the creation and maintenance of software that is specifically tailored to meet the demands of video games [1].

Video games can be implemented using two primary methods: traditional Code-centric Development (CcD) or Model-Driven Development (MDD) [5]. In the case of CcD, game developers manually write and structure the code that is responsible for the game’s behavior, mechanics, and graphics. This approach allows for fine-grained control and customization, but it can be complex and time-consuming. On the other hand, MDD involves creating high-level models that describe the game’s structure, behavior, and components. These models are then transformed into executable code or interpreted at runtime.

Unfortunately, similar to traditional software, the increase in video game complexity is accompanied by an increase in the appearance of software bugs [2]. A good example of this is the blockbuster *Cyberpunk 2077*². Hence, maintenance is becoming more and more important. Bug localization is one of the most important and common activities performed by developers during software maintenance and evolution. Bug localization aims to identify the location in the artifact that is pertinent to a software fault. When models are used for code generation, addressing bugs at the model level must not be neglected [3].

¹T. B. R. Company, Gaming global market report 2023, <https://www.thebusinessresearchcompany.com/report/gaming-global-market-report>.

²How buggy is Cyberpunk 2077, really? - <https://www.pcgamer.com/how-buggy-is-cyberpunk-2077-really/>

While MDD is acknowledged for its advantages in software development [16], particularly in terms of development effort and quality [12], the exploration of bug localization within the MDD context remains an underexplored area.

In this paper, we evaluate whether the development method (MDD or CcD) can have an impact on bug localization tasks by analyzing Kromaia, a commercial video game released on PlayStation 4 and Steam. We present an experiment in which we compare bug localization in MDD and in CcD, in terms of performance, productivity, and satisfaction. A total of 54 subjects (classified as students or professionals) performed the tasks of the experiment, locating bugs in two scenarios of Kromaia. The satisfaction of the subjects was greater when performing bug localization in MDD. However, the performance and productivity results of the students did not show significant differences, while the performance and productivity results of the professionals were better when performing bug localization in CcD.

Our results suggest that there exists a positive perception of MDD. However, the practices used for bug localization need to be evaluated in models due to their suboptimal results. This indicates a knowledge gap in addressing bug localization in models. Given the growing popularity of MDD in the video game domain (and its increasing use in industry-specific domains), there is a need to focus on how engineers perform bug localization in models.

The structure of this paper is as follows. Section 2 reviews the related works in the area. Section 3 presents the case study, Kromaia. Section 4 outlines the experimental design. Section 5 presents the experiment results, followed by a discussion in Section 6. Section 7 summarizes the threats to the validity. Finally, Section 8 concludes the paper.

2.2 Related Work

Although there are no research efforts that empirically compare MDD and CcD from a bug location perspective, previous studies have emphasized the effectiveness of MDD in various domains, showcasing its benefits, especially in small-scale projects under favorable conditions [10, 14]. Existing research has primarily compared MDD and CcD in software development scenarios [12].

Studies by Krogmann and Becker [12], Kapteijns et al. [10], and Mellegård and Staron [14] have provided insights into the efficiency and effectiveness of MDD in different contexts, shedding light on its potential advantages and the distribution of effort between models and other artifacts. Martínez et al. [13]

explored the perceptions of undergraduate students regarding Model-Driven, Model-Based, and Code-centric methodologies, indicating that the Model-Driven approach was considered the most beneficial, although perceived as less aligned with developers' prior experiences. Pappoti et al. [18] further underscored the advantages of MDD, demonstrating shorter development times and fewer challenges when utilizing code generation.

Panach et al. [17] and Domingo et al. [9] delved into the contextual factors influencing the benefits of MDD. Panach et al. affirm its consistent delivery of higher-quality results. Chueca et al. [6] performed an empirical evaluation in the field of game development. They highlighted the enhanced correctness achieved through Software Product Lines (SPLs) based on MDD compared to other traditional methods.

The field of Game Software Engineering (GSE) is gradually becoming a focus, although studies in this specific area still need to be completed. Our work aims to extend the comparative analysis between MDD and CcD by focusing on bug localization in the context of a video game company.

2.3 Background

The case study that we used to assess this experiment focuses on the video game Kromaia³. It is a commercial video game that is accessible on both PC and PlayStation 4 platforms. It immerses players in a three-dimensional environment that combines elements of space exploration, action, and shooting genres.

The developers of Kromaia can follow the principles of MDD. They can employ a Domain-Specific Language (DSL) called Shooter Definition Modeling Language (SDML) to define the attributes and behaviors of the game's entities. The game's entities include bosses, environmental elements, weapons, defenses, and movement behaviors. SDML, which was initially designed for shooter-style video games, has proven to be adaptable for games with similar themes outside of this genre. A simplified subset of the SDML metamodel is shown in the central part of Fig. 3.3. The complete metamodel comprises more than 20 concepts, 20 relationships, and over 60 properties. SDML models are interpreted at runtime to generate the corresponding game's entities. In other words, software models created using SDML are translated into C++ objects at runtime using an interpreter integrated into the game engine [5]. For more

³For more information about Kromaia, you can view the official trailer released by Playstation: <https://youtu.be/EhsejJBp8Go>

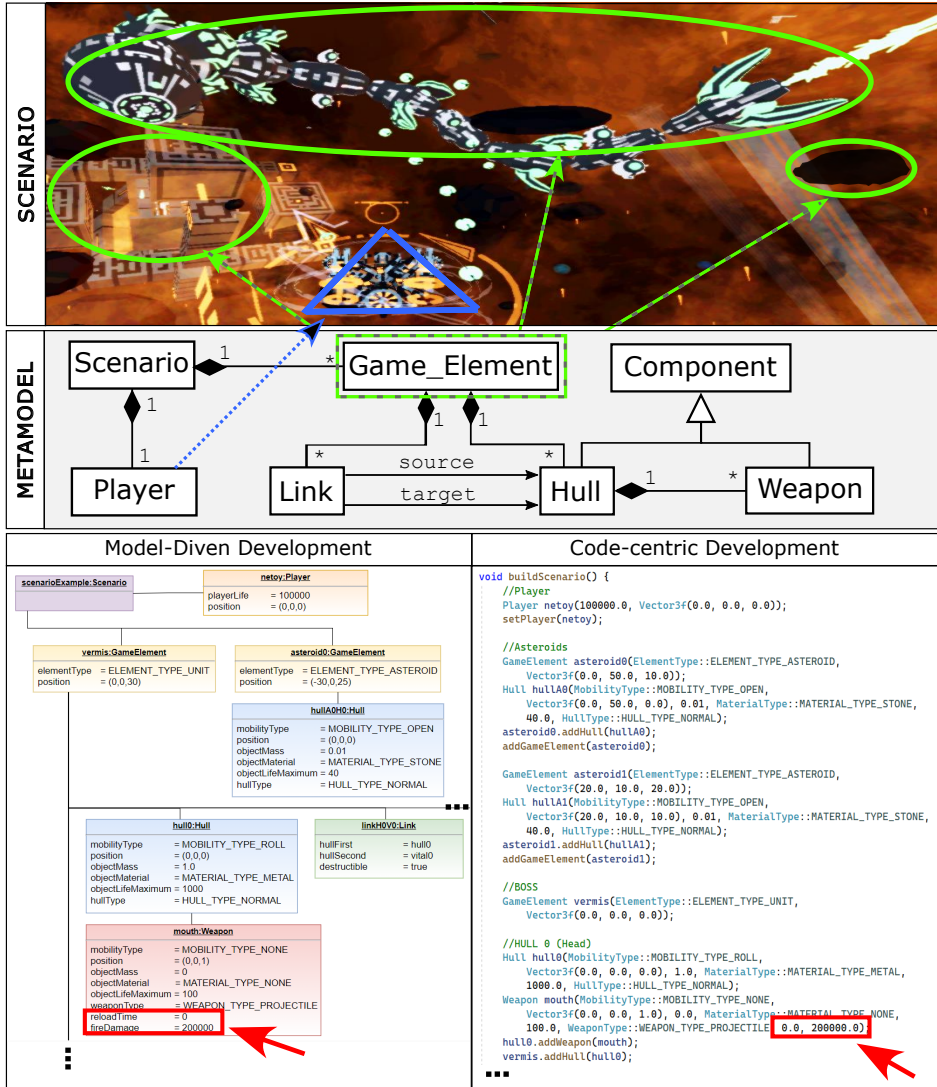


Figure 2.1: Scenario Metamodel and Bug example in MDD and CcD.

information on the SDML model shown in Fig. 3.3, a video presentation can be found here: <https://youtu.be/Vp3Zt4qXkoY>.

SDML allows the development of any game element, such as bosses, enemies, or environmental elements. An actual example of the scenario of a final boss

of Kromaia is presented in the upper part of Fig. 3.3. The Serpent is the final boss that the player must defeat in order to complete Level 1. This boss is composed of one main hull (the head of the serpent) followed by eight hulls. The head has one weapon, and the tail (last hull) has three spike weapons.

In addition, Kromaia developers can also follow a CcD method. They can use the C++ programming language to create content for the game. Game elements created in C++ must have the same rules and structural patterns as those created with SDML in order to fit correctly with the rest of the game.

An actual example of a bug is illustrated in the bottom part of Fig. 3.3. The left part of the figure shows the bug in MDD, while the right part shows the same bug in CcD. In this particular case, a player reported: *"I am unable to fight the final boss of the first level, which is supposed to be the easiest. I die after being hit by a single shot, so I believe that something is wrong"*. A closer inspection reveals that the issue is attributed to an improper weapon configuration. On the one hand, the *fireDamage* attribute exceeds the player's life, resulting in the player's immediate death when attacked by the boss. On the other hand, the *reloadTime* attribute is set to zero, causing the boss to launch continuous attacks without pause. This bug significantly impacts the player experience.

2.4 Experiment Design

2.4.1 Objectives

According to Wohlin's guidelines [21] for reporting software engineering experiments, we have organized our research objectives using the Goal Question Metric template for goal definition [4]. Our goal is to **analyze** software development methods, MDD and CcD, **for the purpose of** comparison, **with respect to** performance, productivity, and user satisfaction, **from the point of view of** students and professionals developers, **in the context of** bug localization for a video game company.

2.4.2 Variables

In this study, the factor under investigation is the software development method (*Method*) in which the bug localization is performed. There are two alternatives: MDD or CcD, which are the methods used by the subjects to perform the bug localization tasks (BL tasks).

Since the goal of this experiment is to evaluate the effects of using different methods when performing bug localization in a scenario of a commercial video game, we selected *Performance* and *Productivity* as the objective dependent variables.

We evaluated *Performance* by calculating precision, recall, and F-measure based on the confusion matrix between subjects' solutions and the ground truth. Precision reflects the accuracy of the identified bugs, recall captures how well all bugs were found, and F-measure balances these two aspects. To represent the *Performance* of the subjects in each BL task, we used the F-measure as the percentage of a BL task solved by a subject. We measured the time used by each subject to finish each task to calculate *Productivity* as the ratio of *Performance* to time spent (in minutes) to perform a BL task.

We also analyzed the methods with respect to *Satisfaction* using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [15]. We use TAM adapted to bug location since it is the questionnaire previously used to measure variables related to satisfaction (comparing MDD and CcD)[8]. We decompose *Satisfaction* into three subjective dependent variables as follows: *Perceived Ease of Use* (PEOU), the degree to which a person believes that learning and using a particular method would require less effort. *Perceived Usefulness* (PU), the degree to which a person believes that using a particular method will increase performance, and *Intention to Use* (ITU), the degree to which a person intends to use a method when performing BL tasks. Each of these variables corresponds to specific items in the TAM questionnaire. We average the scores obtained for these items to obtain the value for each one of these variables.

2.4.3 Design

We chose a factorial crossover design with two periods using two different tasks, T1 and T2, one for each period. The subjects were randomly divided into two groups (G1 and G2). In the first period of the experiment, all of the subjects solved T1 with G1 using MDD and G2 using CcD. Afterwards, in the second period, all of the subjects solved T2 with G1 using the CcD and G2 using MDD.

This repeated measure design enhances the experiment's sensitivity, as noted by Vegas et al. [19]. By observing the same subject using both alternatives, between-subject differences are controlled, thus improving the experiment's robustness regarding variation among subjects. By using two different sequences (G1 used MDD first and CcD afterwards, and G2 used CcD first and MDD afterwards) and different tasks, the design counterbalances some of the effects caused by using the alternatives of the factor in a specific order (i.e., learning effect, fatigue). The effects of the factors period (Task), sequence (Group), and subject will be studied to guarantee the validity of this experiment.

To verify the experiment design, we conducted a pilot study with two subjects. The pilot study facilitated an estimate of the time required to complete the tasks and questionnaires, the identification of typographical and semantic errors, and the testing of the online environment used to create the experiment.

2.4.4 Research Questions and Hypotheses

The research questions and null hypotheses are formulated as follows:

RQ1 - Does the **Method** impact the *Performance* of the BL tasks? The corresponding null hypothesis is $H_{0,Per}$: The **Method** does not have an effect on the *Performance* of the BL tasks.

RQ2 - Does the **Method** impact the *Productivity* of the BL tasks? The corresponding null hypothesis is $H_{0,Pro}$: The **Method** does not have an effect on the *Productivity* of the BL tasks.

RQ3 - Does user satisfaction differ when developers use a different **Method** to solve BL tasks? To answer this question, we formulated three hypotheses based on the variables *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*, with their corresponding null hypotheses. These are: $H_{0,PEOU}$, the **Method** does not have an effect on *Perceived Ease of Use*; $H_{0,PU}$, the **Method**

does not have an effect on *Perceived Usefulness*; $H_{0,ITU}$, the **Method** does not have an effect on *Intention to Use*.

The hypotheses are formulated as two-tailed hypotheses since we have not found empirical studies that support a specific direction for the effect in the video game domain.

2.4.5 Participants

We selected the subjects using convenience sampling [21]. A total of 58 subjects with different knowledge about programming, modeling, and video game development performed the experiment, but only 54 decided to submit their answers and confirmed their agreement to be part of this study. In this study, the participants included 10 professionals working in video game development and 44 third-year undergraduate students who are taking a course in object-oriented programming from a technology program at Universidad San Jorge.

The experiment was conducted by two instructors. During the experiment, one of the instructors gave instructions and managed the focus groups. Both instructors clarified doubts and took notes during the experiment.

2.4.6 Experimental Objects

The tasks of our experiment were extracted from real-world software development, Kromaia [5]. The tasks consisted of localizing two bugs using textual descriptions of the bugs. Each bug was located in a scenario of Kromaia.

To solve the BL tasks using MDD, the subjects used UML diagrams of Kromaia scenarios and the metamodel. To solve the BL tasks using CcD, the subjects used C++ code of Kromaia scenarios and the files of the C++ classes.

A video game engineer who was involved in the development of Kromaia and a researcher designed the two tasks of similar difficulty and prepared the correction templates. In both CcD and MDD, the two tasks consisted of localizing the six error points of a bug given its description. The scenarios where the subjects had to localize the bug had a total of 223 possible error points in Task 1 and 200 in Task 2.

For data collection, we prepared two forms using Microsoft Forms (one for each experimental sequence) with the following sections:

- I An informed consent form that the subjects must review and accept voluntarily. It clearly explains what the experiment consists of and that the personal data will not be collected.
- II A demographic questionnaire was used to characterize the sample.
- III A specific questionnaire for each sequence to collect the subjects' responses during the experiment (their solutions to the tasks, the start and end time of each task, and their answers to the satisfaction questionnaire).

The experimental objects used in this experiment (the training material, the tasks, the correction templates, and the forms used for the questionnaires), as well as the results and the statistical analysis, are available as a replication package at <http://svit.usj.es/MDDvsCcD-BugLocalization>.

2.4.7 Experimental Procedure

The experiment was conducted in two different sessions. The experiment was conducted face-to-face with a group of students in the first session. In the second session, the experiment was conducted online with professionals. During the online session, all of the participants joined the same video conference via Microsoft Teams, and the chat session was used to share information or clarify doubts. The experiment was scheduled to last for one hour and 45 minutes and was conducted following the experimental procedure described as follows:

1. An instructor explained the parts of the session and clarified that the experiment was not a test of the subjects' abilities. (5 min)
2. The subjects attended a video tutorial on the different methods used in Kromaia to develop the video game scenarios and the BL tasks in those scenarios. The information used in the experiment was available to the subjects. (10 min)
3. The subjects received clear instructions on where to find the links to access the forms for participating in the experiment. They were also told about the structure of these forms and where they could find information about the methods used in the experiment. The subjects were randomly divided into two groups (G1 and G2). (5 min)
4. The subjects accessed the online form, and they read and confirmed having read the information about the experiment, the data treatment of

their personal information, and the voluntary nature of their participation before accessing the questionnaires and tasks of the experiment. (5 min)

5. The subjects completed a demographic questionnaire. (5 min)
6. The subjects performed the first task. The subjects from G1 had to use MDD to perform a BL task, and the subjects from G2 had to perform the same task but using CcD. After submitting their solution, the subjects completed a satisfaction questionnaire about the method used for bug localization. (maximum of 30 min)
7. The subjects performed the second task. The subjects from G1 used CcD, and the subjects from G2 used MDD. Then, the subjects completed the satisfaction questionnaire. (maximum of 30 min)
8. One instructor conducted a focus group interview about the tasks while the other instructor took notes. (10 to 15 minutes)
9. Finally, the tasks were corrected, and a researcher analyzed the results.

2.4.8 Analysis Procedure

We have chosen the Linear Mixed Model (LMM) [20] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows us to study the effects of factors that intervene in a crossover design (period, sequence, or subject) and the effects of other blocking variables (e.g., in our experiment, professional experience) [19]. In the hypothesis testing, we applied the Type III test of fixed effects with unstructured repeated covariance. This test enables LMM to produce the exact F-values and p-values for each dependent variable and each fixed factor.

In this study, **Method** was defined as a fixed-repeated factor to identify the differences between using MDD or CcD, and the subjects were defined as a random factor (1|*Subj*) to reflect the repeated measures design. The dependent variables (DV) for this test were *Performance* and *Productivity*, and the three other variables correspond to *Satisfaction*: *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to Use* (ITU).

In order to take into account the potential effects of factors that intervene in a crossover design in determining the main effect of **Method**, we considered **Period** and **Sequence** to be fixed effects. In order to explore the potential effects of the subject's experience to determine the variability in the depen-

dent variables in the statistical model, we also considered the fixed factors **Experience** and the combination of factors **Method** and **Experience**.

We tested different statistical models in order to find out which factors, in addition to **Method**, could best explain the changes in the dependent variables. Some of these statistical models are described mathematically in Formula 2.1. The starting statistical model (*Model 0*) reflects the main factor used in this experiment, **Method** and the random factor ($1|Subj$). We also tested other statistical models (e.g., *Model 1*, *Model 2*, and *Model 3*) that included the fixed factors **Experience** (Exp.), **Period** (Per.) or **Sequence** (Seq.), which could have effects on the dependent variables.

$$\begin{aligned}
 (Model\ 0) \quad DV &\sim Method + (1|Subj.) \\
 (Model\ 1) \quad DV &\sim Method + Exp. + Method * Exp. + (1|Subj.) \\
 (Model\ 2) \quad DV &\sim Method + Seq. + Per. + (1|Subj.) \\
 (Model\ 3) \quad DV &\sim Method + Exp + Method * Exp. + Seq. + Per. + (1|Subj.)
 \end{aligned} \tag{2.1}$$

The statistical model fit of the tested models for each variable was evaluated based on goodness of fit measures such as Akaike's information criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC). The model with the smallest AIC or BIC is considered to be the best fitting model [11, 8].

The assumption for applying LMM is the normality of the residuals of the dependent variables. To verify this normality, we used Shapiro-Wilk tests as well as visual inspections of the histograms and normal Q-Q plots. To describe the changes in each dependent variable, we selected the statistical model that satisfied the normality of residuals and also obtained the smallest AIC or BIC value.

To quantify the differences in the dependent variables due to the factors considered, we calculated the Cohen d value [7], which is the standardized difference between the means of the dependent variables for each factor alternative. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect and values greater than 0.8 indicate a large effect. We selected box plots to describe the results graphically.

2.5 Results

The **Method** factor produced changes in all of the dependent variables. However, its effect size was different between the objective and subjective dependent variables, and, for objective variables, it depended on experience. Table 2.1 shows the values for the mean and standard deviation of the dependent variables for each method and the corresponding Cohen d value measuring the effect size of **Method** for each variable. Positive values of Cohen d value indicate differences in favor of MDD, and negative values indicate differences in favor of CcD. Values indicating medium and high effects are shaded in light and dark gray, respectively.

Table 2.1: Values for the mean and standard deviation of the response variables and Cohen d values for the alternatives of Method (MDD and CcD)

		<i>Performance</i>	<i>Productivity</i>	<i>Satisfaction</i> $\mu \pm \sigma$		
		$\mu \ \% \pm \sigma$	$\mu \ \% / \text{min} \pm \sigma$	<i>PEOU</i>	<i>PU</i>	<i>ITU</i>
All Subjects	MDD	34.66 $\pm$ 27.17	1.88 $\pm$ 1.57	3.45 $\pm$ 0.93	3.39 $\pm$ 0.92	3.41 $\pm$ 1.17
	CcD	33.28 $\pm$ 26.15	1.72 $\pm$ 1.63	2.64 $\pm$ 0.89	2.57 $\pm$ 0.85	2.52 $\pm$ 1.11
	<i>Cohen d</i>	0.052	0.101	0.89	0.932	0.78
Students	MDD	37.64 $\pm$ 26.62	2.02 $\pm$ 1.55	3.53 $\pm$ 0.88	3.49 $\pm$ 0.89	3.57 $\pm$ 1.17
	CcD	31.94 $\pm$ 26.69	1.59 $\pm$ 1.53	2.72 $\pm$ 0.9	2.64 $\pm$ 0.86	2.63 $\pm$ 1.14
	<i>Cohen d</i>	0.214	0.282	0.908	0.971	0.818
Professionals	MDD	21.57 $\pm$ 27	1.24 $\pm$ 1.57	3.1 $\pm$ 1.09	2.99 $\pm$ 0.99	2.7 $\pm$ 0.92
	CcD	39.22 $\pm$ 23.99	2.27 $\pm$ 2.01	2.3 $\pm$ 0.76	2.28 $\pm$ 0.76	2.05 $\pm$ 0.86
	<i>Cohen d</i>	-0.691	-0.575	0.849	0.809	0.729

According to Cohen d values, we can state that when considering all of the subjects who participated in the experiment, the effect size of the **Method** factor in favor of MDD for *Performance* and *Productivity* was negligible, with Cohen d values of less than 0.2 in favor of MDD. When considering only the students, the Cohen d value of 0.21 indicates small effects in favor of MDD, whereas, for professionals, the negative Cohen d value smaller than -0.5 indicates medium effects in favor of CcD. On the other hand, when considering all of the subjects or distinguishing them by experience, the effect size of **Method** in favor of MDD was large, with Cohen d values around 0.8 for the variables related to satisfaction: *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to Use* (ITU). The box plots in Fig. 2.2 illustrate these results.

All of the statistical models shown in Formula 2.1 verify the normality of the residuals. The statistical model that best explains the changes in the *Performance* and *Productivity* variables is *Model 3* of Formula 2.1 since it is the one that obtained the lowest values for the AIC and BIC fit measures. *Model 1* of Formula 2.1 is the one that best explains the changes for the variables related to satisfaction. Table 2.2 shows the results of the Type III fixed effects test for each of the dependent variables and for each fixed factor of the statistical model used in each case. Values indicating significant differences are shaded in grey.

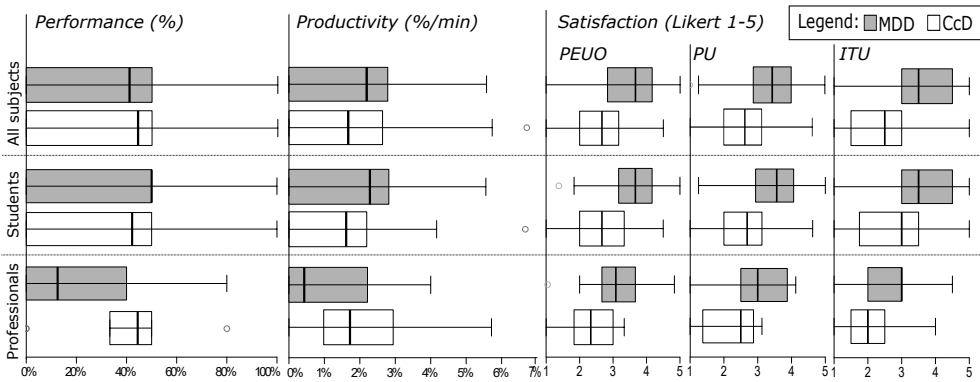


Figure 2.2: Boxplots for dependent variables

Table 2.2: Results of Type III test of fixed effects for each variable and factor. NA=Not Applicable

	Method	Experience	Method*Exp.	Sequence	Period
<i>Performance</i>	F=.890;p=.350	F=.545;p=.464	F=4.416;p=.041	F=5.649;p=.021	F=6.023;p=.018
<i>Productivity</i>	F=.821;p=.369	F=.044;p=.834	F=5.357;p=.025	F=7.103;p=.010	F=.666;p=.418
<i>PEOU</i>	F=24.656;p<.001	F=2.456;p=.123	F=.001;p=.974	NA	NA
<i>PU</i>	F=22.731;p<.001	F=2.760;p=.103	F=.172;p=.680	NA	NA
<i>ITU</i>	F=7.620;p=.008	F=7.620;p=.008	F=.258;p=.614	NA	NA

For *Performance* and *Productivity*, the factor **Method** obtained p-values that are greater than 0.05. Therefore, our first two null hypotheses are not rejected, at least when considering the aggregated data of students and professionals. The p-value of less than 0.05 for the combination of factors **Method*Experience** for *Performance* and *Productivity* indicates that, for both dependent variables, the effect of **Method** on students is significantly different from the effect of **Method** on professionals. By testing the results of the **Experience** separately, the **Method** factor obtained p-values of less than 0.05. However, for students, the differences due to the **Method** factor are significant only when

using *Model 0* of Formula 2.1. For students, the effects of **Sequence** or **Period** are larger than the effects of **Method**. Thus, a comprehensive answer to *RQ1* and to *RQ2* is that there are significant changes in *Performance* and *Productivity* due to the **Method** factor, although the direction and size of these changes vary depending on the subjects' experience, as shown by the descriptive statistics and Cohen *d* values.

On the other hand, the **Experience** factor obtained a p-value greater than 0.05 for *Performance* and *Productivity*, which indicates that there are no significant differences between the average results achieved by students and professionals on BL tasks. Nevertheless, as the **Method*Ex-** **perience** factor indicates, these results are distributed in different ways between the different methods depending on experience. In addition, for *Performance*, the p-values of less than 0.05 obtained by the factors **Sequence** and **Period** indicate significant changes in this variable due to these two design factors. We obtained a Cohen *d* value of 0.474 between the two alternatives of **Sequence** (G1: MDD-CcD, G2: CcD-MDD), indicating medium differences in favor of G1. The subjects who began the experiment by locating bugs in MDD demonstrated better performance results than those who started with CcD. Similarly, the Cohen *d* value of 0.405 between the two alternatives of **Period** (Task 1, Task 2) indicates medium differences in favor of Task 1. The subjects performed better on the first task than on the second task. For *Productivity*, the p-values obtained by the factor **Sequence** are also lower than 0.5, indicating significant changes in the subjects' productivity depending on the method they use first during the experiment. The Cohen *d* value of 0.580 between the two alternatives of **Sequence** indicates medium differences in favor of G1. The subjects who started the experiment solving the BL task in MDD were more productive in both tasks than those who started the experiment solving the BL task in CcD. The differences that we observed in the subjects' performance between the first task and the second task were not reproduced in their productivity since the subjects spent more time on the first task than on the second.

For the variables related to satisfaction, *Perceived Ease of Use*, *Perceived Usefulness* and *Intention to Use*, the **Method** factor obtained p-values of less than 0.05. Therefore, our third null hypothesis is rejected, and the answer to *RQ3* is affirmative: The user satisfaction is significantly different when developers use a different **Method** to solve BL tasks. **Method** has significant large effects on *PEOU*, *PU*, and *ITU*, which confirms the statistical significance of the differences observed in the mean values of the variables related to satisfaction. In addition, the factor **Experience** obtained a p-value of less than 0.05 for *ITU*. The Cohen *d* value of 0.606 between students and professionals indicates

a medium effect in favor of the students. The students gave higher scores than the professionals on *ITU* for both of the methods used in the experiment.

2.6 Discussion

Overall, both students and professionals reported greater satisfaction when using MDD for bug localization. However, the results of performance and productivity do not show significant differences when considering the group of students. In contrast, when considering the group of professionals, the results of performance and productivity show significant differences in favor of CcD. The results of satisfaction and the focus group comments suggest a positive perception of models in this context, although the results of performance and productivity are not aligned with the subjects' satisfaction.

This observation is consistent with the current state of the scientific literature in this field. Many research efforts make claims regarding the benefits of models, including bug reduction. Nevertheless, the effectiveness of bug localization in models has not been exhaustively evaluated, perhaps because of the assumption that it would be better to use models.

During the focus group, some subjects highlighted that having performed the location in the model before helped them to understand the C++ code. This led to improved performance and productivity in the bug localization tasks for those who had performed the MDD-CcD sequence. These outcomes are supported by the results and hypothesis tests. In addition, the students stated that they preferred to perform bug localization in the models. However, professionals who have a greater proficiency in software development prefer to perform bug localization in the C++ code.

It seems that in CcD, the subjects have a way of navigating (e.g., starting from a method relevant to the bug, they inspect the methods that call it or the methods it calls). However, in MDD, when a model element relevant to the bug is located, it is not clear to the subjects how to continue inspecting the model. One may think that graphically connected model elements (e.g. connected with relationships) are also relevant to the bug, but the bug may involve unconnected model elements. This suggests that it is not straightforward to transfer bug localization in CcD to bug localization in MDD.

Our findings suggest that there is a lack of knowledge on how to approach bug localization in models. This might mean that the established practices yield suboptimal results when they are applied to models. In fact, the students

who are more unfamiliar with established practices in software development obtained better results when performing bug localization in models. In contrast to the professionals, the students used more intensively the metamodel to locate bugs, which can also explain why their results were better.

Considering the growing popularity of MDD in the video game domain and their increasing use in industry-specific domains, there is a critical need to focus on how engineers perform bug localization in models. It is important to move away from the current trend of assuming that models are inherently superior for bug localization, as empirical evaluations do not support this assumption.

2.7 Threats to Validity

To describe the threats to validity of our work, we use the classification of Wohlin et al. [21]. This section shows the threats that affected the experiment.

Conclusion validity: The *low statistical power* was minimized because the confidence interval is 95%. To minimize the *fishing and the error rate* threat, the statistical analysis has been done by a researcher who did not participate in the task design or in the correction process. The *reliability of measures* threat was mitigated because the measurements were obtained from the data sheets that were automatically generated by the forms with the answers of the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the procedure was identical in the two sessions. Also, the tasks were designed with similar difficulty.

Internal validity: To avoid the *instrumentation* threat, we conducted a pilot study to verify the design and the instrumentation. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience, different levels of modeling or coding, and different levels of knowledge of the video game domain. To mitigate this threat, the treatment was applied randomly.

This threat also affected the experiment because of the voluntary nature of participation. We selected students from a course whose content was in line with the experiment activities to avoid student demotivation.

Construct validity: To mitigate the *mono-method bias* threat, we mechanized the measurements as much as possible by means of correction templates. To weaken the *evaluation apprehension* threat, at the beginning of the experiment, the instructor explained to the subjects that the experiment was not

a test of their abilities. The instructor also told the students that neither participation nor results would affect their grades in the course where the experiment took place. In order to mitigate the *author bias* threat, the tasks were extracted from a commercial video game and were designed by the same experts with similar difficulty for the two methods compared. The experiment was affected by the *mono-operation bias* threat since we worked with only two BL tasks from a specific video game.

External validity: The *interaction of selection and treatment* threat affects the experiment because it involves a larger number of students than professionals, making students more represented in the overall results than professionals. The *domain* threat occurs because the experiment has been conducted in a specific domain (video game) and for a very specific type of task, i.e., to solve a BL task in a scenario of Kromaia video game. We think that other experiments in different games should be performed to validate our findings.

2.8 Conclusion

In this work, we present an experiment that compares MDD and CcD in terms of performance, productivity, and satisfaction. This work investigates the influence of MDD and CcD on bug localization, employing the video game Kromaia (a commercial video game) as a case study. Bug localization in MDD increased satisfaction, but professionals performed better in CcD. The subjects' techniques that were applied for bug localization showed suboptimal performance in models, highlighting a knowledge gap in MDD bug localization. With the growing popularity of MDD in video games, exploring effective bug localization methods specific to MDD is essential.

Bibliography

- [1] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2010.05.004> (cit. on p. 64).
- [2] Lorena Arcega, Jaime Font, and Carlos Cetina. “Evolutionary Algorithm for Bug Localization in the Reconfigurations of Models at Runtime”. In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. MODELS '18. Copenhagen,

- Denmark: Association for Computing Machinery, 2018, 90–100. ISBN: 9781450349499. DOI: 10.1145/3239372.3239392 (cit. on p. 64).
- [3] Lorena Arcega et al. “An approach for bug localization in models using two levels: model and metamodel”. In: *Software and Systems Modeling* 18.6 (2019), pp. 3551–3576. DOI: 10.1007/s10270-019-00727-y (cit. on p. 64).
 - [4] Victor R. Basili and H. Dieter Rombach. “The TAME Project: Towards Improvement-Oriented Software Environments”. In: *IEEE Transactions on Software Engineering* (1988). ISSN: 00985589 (cit. on p. 68).
 - [5] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *Journal of Systems and Software* 171 (2021), p. 110804 (cit. on pp. 64, 66, 71).
 - [6] Jorge Chueca et al. “Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development”. In: *Journal of Systems and Software* 205 (2023), p. 111824. ISSN: 0164-1212 (cit. on p. 66).
 - [7] Jacob Cohen. “Statistical power for the social sciences”. In: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (cit. on p. 74).
 - [8] África Domingo et al. “Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives”. In: *Advanced Information Systems Engineering*. Springer, 2021, pp. 483–498. ISBN: 978-3-030-79382-1 (cit. on pp. 69, 74).
 - [9] África Domingo et al. “Evaluating the Benefits of Model-Driven Development”. In: *Advanced Information Systems Engineering*. Springer, 2020, pp. 353–367 (cit. on p. 66).
 - [10] T Kapteijns et al. “A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare”. In: *From code centric to model centric software engineering Practices Implications and ROI* (2009) (cit. on p. 65).

- [11] Evrim Itir Karac, Burak Turhan, and Natalia Juristo. “A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development”. In: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589 (cit. on p. 74).
- [12] K Krogmann and S Becker. “A Case Study on Model-Driven and Conventional Software Development : The Palladio Editor”. In: *Software Engineering* (2007) (cit. on p. 65).
- [13] Yulkeidi Martínez, Cristina Cachero, and Santiago Meliá. “MDD vs. traditional software development: A practitioner’s subjective perspective”. In: *Information and Soft. Technology*. 2013 (cit. on p. 65).
- [14] Niklas Mellegård and Mirosław Staron. “Distribution of effort among software development artefacts: An initial case study”. In: *LN in Business Information Processing*. 2010. ISBN: 9783642130502 (cit. on p. 65).
- [15] Daniel L Moody. “The method evaluation model: a theoretical model for validating information systems design methods”. In: *ECIS 2003 proceedings* (2003), p. 79 (cit. on p. 69).
- [16] Gunter Mussbacher et al. “The relevance of model-driven engineering thirty years from now”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2014, pp. 183–200 (cit. on p. 65).
- [17] Jose Ignacio Panach et al. “In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction”. In: *Information and Software Technology* (2015). ISSN: 09505849 (cit. on p. 66).
- [18] Paulo Eduardo Papotti et al. “A quantitative analysis of model -driven code generation through software experimentation”. In: *Advanced Information Systems Engineering*. Springer. 2013, pp. 321–337 (cit. on p. 66).
- [19] Sira Vegas, Cecilia Apa, and Natalia Juristo. “Crossover designs in software engineering experiments: Benefits and perils”. In: *IEEE Transactions on Soft. Eng.* 42.2 (2015), pp. 120–135 (cit. on pp. 70, 73).

- [20] Brady T West, Kathleen B Welch, and Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman and Hall/CRC, 2014 (cit. on p. 73).
- [21] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 68, 71, 79).

Co-evolving Scenarios and Simulated Players to Locate Bugs that arise from the Interaction of Software Models of Video Games

Game Software Engineering (GSE) is a field that focuses on developing and maintaining the software part of video games. A key component of video game development is the utilization of game engines, with many engines using software models to capture various aspects of the game. A challenge that GSE faces is the localization of bugs, mainly when working with large and intricate software models. Additionally, the interaction between software models (i.e. bosses, enemies, or environmental elements) during gameplay is often a significant source of bugs. In response to this challenge, we propose a co-evolution approach for bug localization in the software models of video games, called CoEBA. The CoEBA approach leverages Search-Based Software Engineering (SBSE) techniques to locate bugs in software models while considering their interactions. We conducted an evaluation in which we applied our approach to a commercial video game, Kromaia. We compared our approach with a state-of-the-art baseline approach that relied on the bug localization approach used by Kromaia's developers and a random search used as a sanity check. Our co-evolution approach outperforms the baseline approach in precision, recall, and F-measure. In addition, to provide evidence of the significance of our results, we conducted a statistical analysis that shows significant differences in precision and recall values. The proposed approach, CoEBA, which considers the interaction between software models, can identify and locate bugs that other bug localization approaches may have overlooked.

Author version of the article: I. Roca, O. Pastor, C. Cetina, and L. Arcega, “Co-evolving Scenarios and Simulated Players to Locate Bugs that arise from the Interaction of Software Models of Video Games”. In: Information and Software Technology, Volume 169, Page 107412, doi: 10.1016/j.infsof.2024.107412.

3.1 Introduction

The video game industry has become a significant and lucrative software industry sector¹. As video games continue to gain popularity, the development of these games has become increasingly complex, involving the creation of sophisticated software covering multiple aspects of the game, such as game mechanics, graphics, and physics. Game Software Engineering (GSE), therefore, has emerged as a field that focuses on the development and maintenance of software for video games [3].

In video game development, game engines serve as a basic resource that combines elements such as graphics and physics engines with a suite of accompanying tools. While there are popular game engines [32, 19], some studios opt for custom solutions tailored to their specific needs [17]. A recent literature review [34] identifies both widely used game engines such as Unity or Unreal and specific engines developed by the video game studios themselves. A key asset of game engines is software models. In the scope of this work, the concept of software model should not be confused with the concept model used in computer graphics and video games to refer to the visual representation of 3D shapes or geometry. An actual example of a model of the scenario of a final boss of Kromaia is presented in Figure 3.1. In addition, Kromaia developers can also use the C++ programming language to create content for the game. Game engines offer developers a choice between coding, often in languages such as C++, or using software models that are integrated into the engines (as illustrated in Figure 3.2 [14]). These models provide a higher level of abstraction, allowing developers to focus on designing the core content of the game. Typically, game engines incorporate their own modeling languages (for instance, Unreal employs Blueprint models), while recent trends in Model-Driven Game Development [34] highlight the adoption of other languages such as UML and Domain Specific Languages (DSLs).

¹T. B. R. Company, Gaming global market report 2023, <https://www.thebusinessresearchcompany.com/report/gaming-global-market-report> (2023).

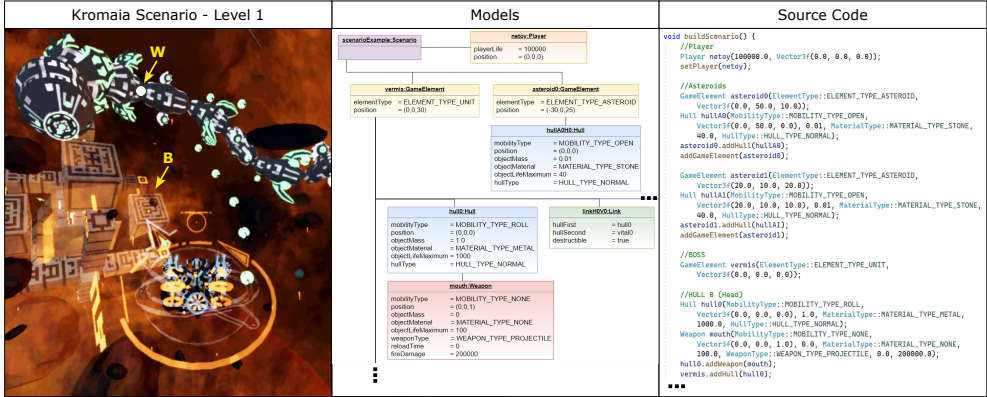


Figure 3.1: Example of a scenario developed using models and code.

This approach, similar to Model-Driven Engineering (MDE), has proven to be useful in simplifying video game development. However, applying MDE to video games requires specific bug localization approaches to ensure the quality of the software. Currently, bug location approaches mainly focus on source code without considering other crucial elements such as software models. This is shown in a survey on bug location [33], which highlights that none of the existing approaches consider models as a potential source of bugs. This lack of appropriate approaches can have a negative impact on video game development. It often results in an increase in development time, which can lead to delays in delivery and, ultimately, the release of games with a high number of bugs. A good example of this is the blockbuster *Cyberpunk 2077*², which was released after nine years of development with numerous bugs. Such was the level of bugs that the game had to be pulled from store shelves, and even a year after its initial release, patches were still being released to fix them.

In this context, it is important to note that video games have different characteristics compared to classic software in both, developing and maintaining, due to the differences in the artifacts used (i.e., shaders, meshes, or prefabs) and the interactive nature of video games. Additionally, game developers face different challenges than non-game developers when locating bugs and reusing code [28].

In video game development, like the boss shown in Figure 1, content can be defined through code or models (see the central part of Figure 1 for the model and

²How buggy is Cyberpunk 2077, really? - <https://www.pcgamer.com/how-buggy-is-cyberpunk-2077-really/>

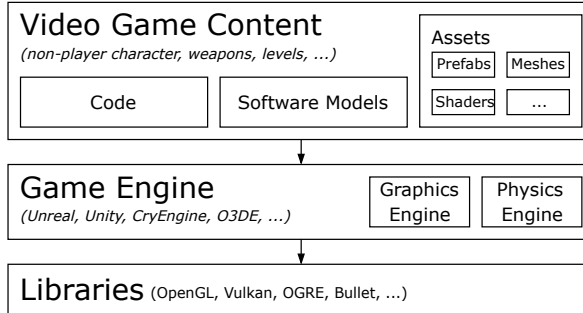


Figure 3.2: Overview of video game artifacts.

the right part for the code). Models offer higher-level concepts and abstraction from implementation details, providing a smaller bug localization search space. However, despite this advantage, existing literature on bug localization in models is scarce, posing a challenge. As models are key artifacts in video game development, bug localizations should be performed on models.

Furthermore, due to the interactive nature of video games, the interaction between software models in the gameplay can be a significant source of bugs and cannot be ignored. For instance, the interaction between the player, the game environment, and the game mechanics can result in unpredictable behavior and bugs. The boss that appears in Figure 1 is a powerful enemy that appears at the end of the level. The bosses have weak points that must be attacked to destroy them. However, a boss can be invincible because a weak point is overlapped. This bug occurs when solid objects overlap each other when they are not supposed to. Letter W in Figure 3.1 indicates a weak point. When the boss hits some blocks (letter B in Figure 3.1) that are part of the scenario, this type of bug occurs. Someone may argue that this bug may occur due to the emergent behavior of the game. However, other bugs can occur even in deterministic games. For example, a boss can have wrong behavior because the movement is blocked. This bug occurs when the parts of the boss are incorrectly positioned. Incorrect positioning blocks the movement of other parts of the model; for example, the position of one part invades the space of the other. If they invade each other the physics have unpredictable behavior. This bug occurs due to an error in the boss’s programming (in code or models). In games that do not provide linear experiences, the player has many possibilities to perform different behaviors. This leads to the concept of emergent gameplay. It describes situations where a game’s mechanics and systems interact in ways that were not intentionally designed by the developers but rather created

through the choices and actions of players. In addition, games can incorporate random elements. Such elements can include factors like randomized enemy behavior, variable item drops, or procedural level generation. Therefore, developers have the ability to create diverse and dynamic experiences for players, making each playthrough unique.

Given the complexity of video games thorough game testing becomes imperative. Game testing involves systematically evaluating the game’s mechanics, interactions, and random elements to identify bugs, balance issues, and unintended emergent behaviors. In addition, games can also have many scenarios, which produce an excessively high amount of different combinations that are unaffordable to be playtested by a human [30].

In this work, we present an approach, called CoEBA, for bug localization in game software engineering that considers the interaction between different software models. Our approach applies Search-Based Software Engineering (SBSE) [21] techniques to locate bugs by exploring the search space evolving software models. Our CoEBA approach is designed to co-evolve simulated players and scenarios to identify potential bugs in software models while considering their interactions. Simulated players represent player behaviors during the gameplay, and scenarios represent the environment around the player including context, enemies, and bosses. In these scenarios is where model interactions occur. In order to validate the effectiveness of our approach, we compared it with two other approaches, a baseline [14] and a random search using the Kromaia case study [11]. Kromaia is a commercial video game that involves flying and shooting with a spaceship in 3D space and has been released on multiple platforms and translated into different languages.

To evaluate the CoEBA approach, we have formulated three research questions:

- *RQ₁*: What is the performance in terms of solution quality of the CoEBA, the baseline, and the random search approaches?
- *RQ₂*: Is there any improvement in performance taking into account model interaction?
- *RQ₃*: Are the differences in performance results obtained by the CoEBA, the baseline, and the random search approaches significant?

In our evaluation, we use widely accepted metrics such as precision, recall, and F-measure to evaluate the performance of our approach, the baseline approach, and a random search approach. Our results show that CoEBA outperformed

the baseline and the random search approaches regarding recall, precision, and F-measure. The precision, recall, and F-measure values achieved by CoEBA are 45.76%, 84.83%, and 53.87%, respectively. We performed a statistical analysis to provide quantitative evidence of the impact of the results. This analysis shows that the impact on performance is significant.

From the evaluations performed, we can say that the interaction between models is relevant to locating bugs. This is because it defines the behavior of the video game as a whole and provides insights to facilitate bug location improving the performance of the approach.

The structure of this paper is as follows. Section 2 provides an overview of the case study (Kromaia) and the motivation. Section 3 describes our proposed approach, CoEBA. In Section 4, we present the evaluation methodology of our approach with Kromaia. In section 5, we present the results. In sections 6 and 7, we discuss the results and analyse the threats to validity respectively. In Section 8, we present a review of the related work in this area. Finally, in Section 9, our conclusions are summarized.

3.2 Background and motivation

This section presents the case study we employed to evaluate this work. The case study involves the video game Kromaia³. Kromaia is a commercial video game released on PC and PlayStation 4. It is set in a three-dimensional space. The game is a blend of space exploration, action, and shooting genres. Each level requires the player to pilot their spaceship from a starting point to a target destination while exploring floating structures, avoiding asteroids, and collecting items along the way. The player must also defeat basic enemies attacking their spaceship by firing projectiles. Once the player reaches the destination, the final boss for that level appears, and the player must defeat the boss to progress to the next level.

The developers of Kromaia use a Domain Specific Language (DSL) called Shooter Definition Modeling Language (SDML) to describe the various characteristics and behaviors of the game's entities, including bosses, environmental elements, weapons, defenses, and movement behaviors. The developers of Kromaia implemented this DSL to define shooter-style video games. However, its applicability extends beyond this genre to other video games with similar themes. This modeling language has a fundamental role in the definition of

³To learn more about Kromaia, you can watch the official trailer released by Playstation: <https://youtu.be/EhsejJBp8Go>

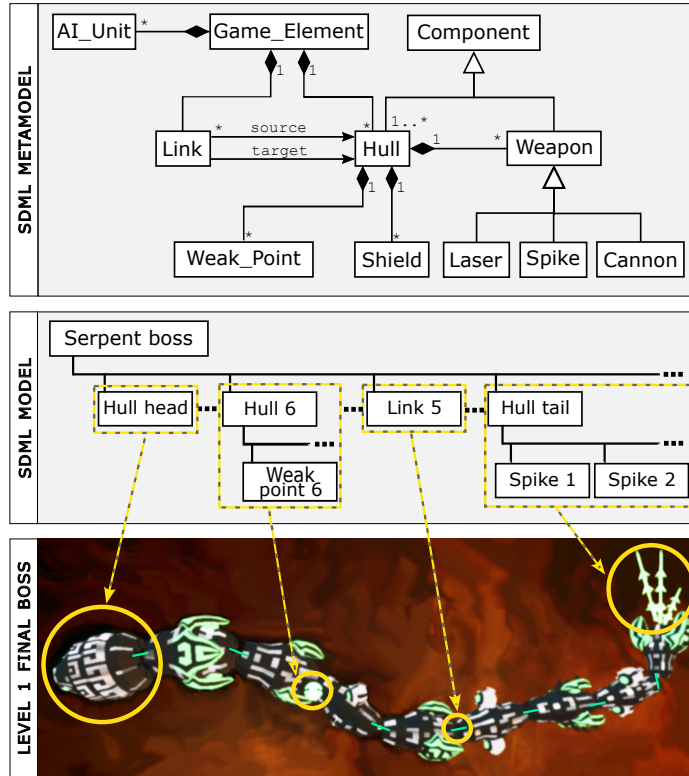


Figure 3.3: The Shooter Definition Modeling Language (SDML).

various attributes in relation to game entities. It provides a comprehensive description of the anatomical structure of the characters, covering details such as the constituent parts, their physical characteristics, and how they are interconnected. In addition, the DSL delves into the configuration of vulnerable sections, weaponry, and protective measures within the character's structure or body. It also regulates the various movement behaviors associated with the character as a whole or its individual components. SDML follows the principles of Model-Driven Engineering (MDE), allowing developers to specify the rules of engagement between entities and the game environment, such as boundaries, obstacles, and environmental hazards. The upper part of Figure 3.3 presents an excerpt version of the SDML metamodel that provides a comprehensive understanding of the game's elements in Kromaia. The complete metamodel consists of over 20 concepts, 20 relationships, and more than 60 properties. The SDML models are interpreted at runtime to generate the corresponding entities in the video game. This means that the software models are created using SDML and translated into C++ objects at runtime by an interpreter used by the game engine. Kromaia was developed with a specific video game engine created by the company [11].

Kromaia's specific engine uses concepts similar to those of widespread engines such as Unity or Unreal. In fact, Kromaia's SDML models can be loaded into Unity's WebGL (<https://svit.usj.es/models22/bl-in-mgse>). This engine acts as a framework in the context of the video game architecture and allows developers to create content in two different ways:

- Programming, using the framework's Application Programming Interface (API).
- Software Models, which are created using SDML and translated to its programming equivalent at run-time by an interpreter that is used by the engine.

Therefore, developers should track only what they develop for each part of the game content. In the specific case of Kromaia, the developers started to create content with code, but to be more productive they switched to SDML models to create the content of the game (e.g., levels, NPCs, items, and weapons). More can be learned about the SDML model of Figure 3.3 in the following video: <https://youtu.be/Vp3Zt4qXkoY>

SDML allows the development of any game element such as bosses, enemies, or environmental elements. An actual example of a final boss of Kromaia is presented in the bottom part of Figure 3.3. The Serpent is the final boss that

the player must defeat to complete level 1. The central part of the figure shows the model of the boss using the SDML. This boss is composed of one main hull (the head of the serpent) followed by eight hulls. The tail (last hull) has two shields and three spike weapons. As in this example, all bosses, enemies, and elements in Kromaia are implemented using SDML.

In order to locate bugs related to the SDML models that define the behavior of the game's bosses, a previous work employs an approach that involves running game simulations [14]. These simulations are designed to simulate a one-on-one battle between a boss and a simulated player that represents the actions and decisions of a human player. The simulated player is powered by an algorithm that the Kromaia team developed. The developers used the algorithm during the game's development to evaluate the content they made. The algorithm focuses on simulating the player, so it is independent of changes in the other elements of the game (attacking weak points and avoiding taking damage or colliding with the scenario). However, the algorithm is game-dependent (space shooter), so it cannot be used as it is in other games. However, it would be possible to find similar algorithms in other games, e.g., those controlling opposing cars in a racing game, enemy generals commanding troops in an RTS game, or bots in an FPS game. The algorithm can be parametrized to define the fighting strategy. The simulated player parameters were provided by the developers based on the analysis of battles between human players and bosses. It considers aspects such as the weaponry used by each of them and the differences between the two entities in terms of agility, speed, endurance, or size. For example, the parameters can define the time that the player spends in each element, how the level is traversed, or the player's reaction when the player is hit. During the simulated battle, both the boss and the simulated player are programmed to strive for victory and avoid tie games at all costs, ensuring that there is a clear winner.

In addition to the bosses and the simulated player, environmental elements in Kromaia play a crucial role in shaping the gameplay experience. These elements, such as walls, spikes, and towers, significantly impact the actions and strategies of both the boss and the player during a duel. The placement and layout of these elements can create opportunities for the boss and the player to gain advantages or disadvantages and influence their movement patterns and attack styles. For example, walls can serve as a strategic cover for the boss, allowing them to evade incoming attacks or gain an advantageous position to launch a counter-attack. Similarly, the player may use walls to dodge or avoid the boss's attacks or launch surprise attacks behind the cover. Towers that

shoot projectiles can provide an additional challenge for both the boss and the player, as they must be avoided or destroyed to gain an advantage.

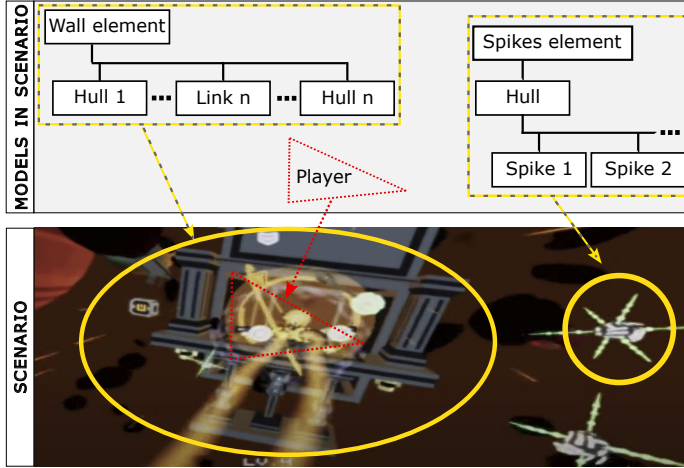


Figure 3.4: Example of model interaction in the scenario.

Moreover, the presence of spikes or other environmental elements can further enrich the gameplay experience, as the player needs to be aware of their location and avoid taking damage from them. The placement of these environmental elements can force the player to alter their attack strategies or movement patterns to avoid taking damage, adding a layer of complexity to the game. Thus, to achieve a realistic simulation between the boss and the simulated player, it is essential to consider the impact of these environmental elements in conjunction. We refer as a scenario to the interaction of all these models (bosses and elements of the environment) that conform to the same metamodel. Figure 3.4 shows an example of a real scenario with walls and spikes and the player in the middle.

The figure depicted in Figure 3.5 presents two different simulations that emulate a player's behavior during a battle with a boss in the game. These simulations aim to predict how the boss and simulated player will behave under different conditions and to locate bugs in the software models of the game. The triangle symbol in the figure represents the simulated player, while the circles and lines that connect them correspond to game elements, such as the boss, towers, spikes, or walls. The dashed lines indicate the path followed by the simulated player or the projectiles, and the crosses represent the attacks performed by the simulated player, the boss, and the environmental elements.

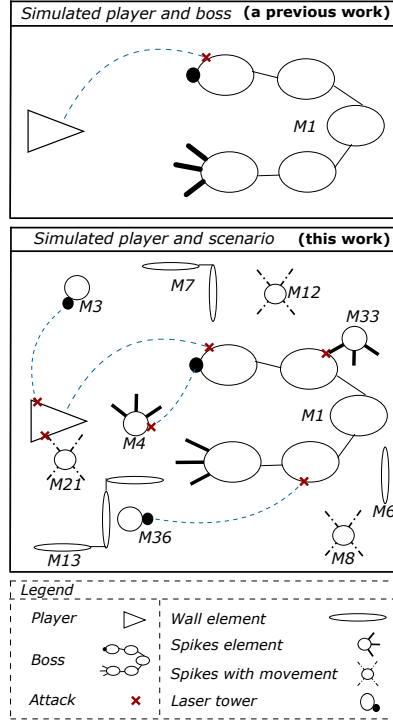


Figure 3.5: Example of two different simulations: simulated player and boss (one software model), simulated player, boss, and environmental elements (several software models).

The upper simulation presented in Figure 3.5 shows a duel between the simulated player and a boss without considering the environmental elements (a previous work [14]). The bottom example shows a simulation between the simulated player, a boss, and (taking into account) the environmental elements (the one used in this work). That is, this simulation considers several software models of the video game (i.e., boss, towers, spikes, or walls) and the interactions that these models have throughout the simulation. The $M1$ model represented in this figure is the level 1 final boss shown in Figure 3.3, the rest of the models ($M3$, $M4$, $M6$, $M7$, $M8$, $M12$, $M13$, $M21$, $M33$, and $M36$) represented show environmental elements that are present in the scenario and that have been defined using SDML (Fig 3.3).

The interaction among the software models of the environmental elements, the boss, and the simulated player can also generate unwanted bugs that may impact the game's overall quality. For instance, if the boss's software model

is not appropriately designed to handle the presence of walls or towers, this could cause unintended behaviors that could lead to glitches and an unpleasant gaming experience for the player. Similarly, if the player's behavior is not adapted to the presence of environmental elements, such as walls or spikes, this could result in unintended collisions or ineffective attacks. To minimize the risk of unwanted bugs, it is crucial to incorporate the impact of environmental elements into the simulations and thoroughly test the gameplay. This can help to ensure that the interactions between the boss, the simulated player, and the scenario are smooth and error-free, leading to a more enjoyable and immersive gaming experience.

3.3 Co-Evolving scenarios and simulated players to locate bugs in the interaction of software models of video games

The gameplay of a video game can involve numerous combinations of scenarios (game elements in the scene) and player strategies, so the number of different model interactions that can be triggered is very high. Since the search space is too large, it is difficult to perform playtesting manually, exploring the space of possibilities exhaustively [30]. To tackle this challenge, we leverage SBSE through a co-evolutionary algorithm to explore the vast search space produced by combinations of scenarios (game elements in the scene) and simulated players (player behaviors in the battle).

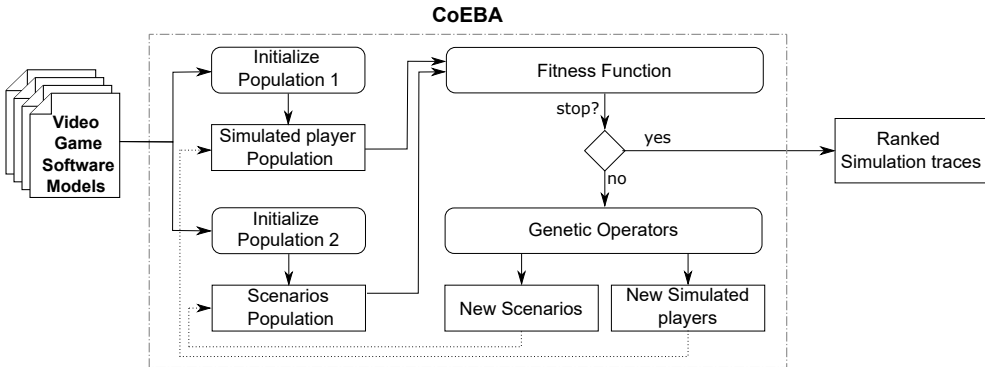


Figure 3.6: CoEBA: Co-Evolutionary Bug Localization approach for GSE.

In co-evolutionary algorithms, instead of evolving a population of individuals representing the solution, they co-evolve subpopulations representing specific parts of the solution. There are two types of co-evolution: cooperation and

competition [25, 12]. We have selected a competitive co-evolutionary algorithm for its ability to solve problems involving interactions between different entities. Such problems arise in situations where entities require adaptation and evolution in response to their opponents' strategies, as in games in which competitive co-evolutionary algorithms excel [13, 18]. Taking into account the problem we want to solve, we follow a competitive approach taking into account the simulated players and the environmental elements where the fitness value of an individual from one subpopulation depends on the fitness value of other individuals from the other subpopulation.

In this section, we first present an overview of our approach and then provide the details of the approach and our adaptation of the co-evolutionary algorithm to deal with the specific challenges of software models of video games.

3.3.1 Overview of our approach

Figure 3.6 presents our Co-Evolutionary Bug Localization approach for GSE called CoEBA. The left part shows the input for the approach: the set of software models that describe the video game's content. The center shows the main steps. The 'Initialize Population' step calculates the two initial populations. The 'Fitness' step assigns values that assess how good each individual of each population is. Finally, the 'Genetic Operations' step produces new generations of both populations. The approach outputs a ranked list of simulation traces that can trigger the bug.

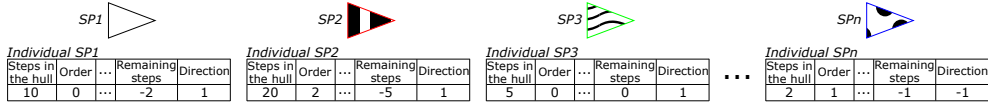
3.3.2 CoEBA populations

CoEBA works with two different populations. The first population is composed of simulated players, which represent player behaviors during battles. The second population is composed of game scenarios, encompassing diverse combinations of game elements within a given scene. To represent each individual of each population, we use a vector representation.

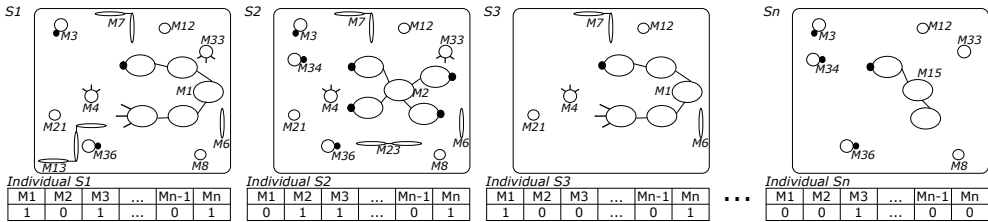
For the first population, each vector's dimension represents a simulated player parameter. The simulated player emulates the behavior of a player in a determined scenario when the battle with the boss occurs. For example, the parameters can define how many steps the simulated player takes in each hull of the boss or each environmental element, the order in which the hulls and the environmental elements are visited following different patterns, the behavior of the player when they are attacked, or the direction used to visit the hulls of the boss or the environmental elements. Thus, an individual is defined as

a set of parameters applied to a simulated player. The size of the individual corresponds to the number of parameters (dimensions) in the vector. The developers provided the simulated player parameters based on the analysis of real games played. Figure 3.7 upper part shows four different simulated players. The SP1 example shows a simulated player that attacks the first hull and then moves away. The SP2 example shows a simulated player that attacks the first hull, then skips one, and attacks the following hulls to the end of the boss.

Population 1 - Simulated players



Population 2 - Scenarios



Simulation

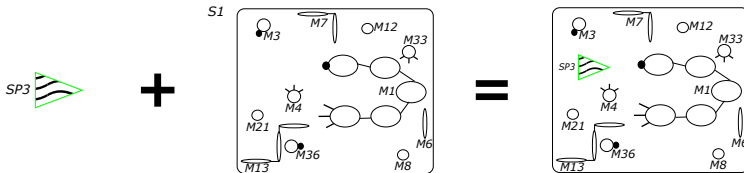


Figure 3.7: Examples of individuals and their representation from both populations simulated players and scenarios, and an example of a simulation (simulated player + scenario)

For the second population, each vector's dimension represents a software model that can appear (or not) in the scenario (see the central part of Figure 3.7). That is, each vector represents the different types of environmental elements (software models) that can come onto the scene. Each vector in the population has a dimension corresponding to a specific scenario and contains information about the software models present in the scenario. The number of different types of environmental elements available in the game determines the number of dimensions in the vector. These environmental elements have been identified and defined by the Kromaia developers, and we have used their specifications to create our vector representation for the second population.

The central part of Figure 3.7 presents four different scenarios. Each example presents different types of software models that appear as environmental elements in the scenario. To represent the S1 example, the individual would have the dimension corresponding to $M3$ set to 1, while the dimension corresponding to $M2$ would be set to 0. In contrast, the S2 example would have the dimension corresponding to $M2$ set to 1, while the dimension corresponding to $M13$ would be set to 0. The number and the type of these environmental elements can play an essential role while the combination of a simulated player and a scenario (that is, a simulation, see bottom part of Figure 3.7) is running. Hence, the interaction of the software models of the environmental elements, the player, and the boss when all of them are put together in the scenario can trigger unwanted bugs.

3.3.3 CoEBA fitness

The fitness function used by our approach assesses simulated players and scenarios based on their ability to detect game bugs. It is based on two functions for each population: 1) evaluating their alignment with how the game developers intended for the game to behave; 2) maximizing the coverage of the less suitable simulated players or scenarios in the generation.

In both populations, the first function involves a duel between the game's player and boss. The goal is to evaluate how closely the game's behavior aligns with the developers' intent. This function calculates its fitness value based on the percentage of player victories and the percentage of player health remaining after winning a simulated battle. The calculations for victory and health are performed in the same manner as described in previous studies [11, 14]. In a previous study by Blasco et al. [11] (content generation), simulations closest to the developers' intentions were rewarded. However, the opposite approach is taken for bug localization [14]. Hence, in this work, simulated players that deviate the most from what developers intended are rewarded and ranked first in the function.

For the second fitness function, we select the top individuals from each population, which are the worst simulated players and scenarios. For assessing each simulated player, our approach considers the number of scenarios that did not achieve expected victory and health values in the duel. The more scenarios that generate unwanted behavior with the simulated player, the higher the fitness value. In order to normalize the numerical value, we establish this as a percentage of the undesired scenarios against the total number present in the population. Similarly, our approach considers the number of simulated

players that did not achieve expected victory and health values in the duel to assess each scenario. The more simulated players that generate unwanted behavior with the scenario, the higher the fitness value. In order to normalize the numerical value, we establish this as a percentage of the undesired simulated players against the total number present in the population. Thus, resulting values can vary from 0 to 1.

The fitness value of a simulated player is the average value between the first and the second fitness values described above. Subsequently, our algorithm orders simulated players and scenarios in an ascending manner with those having lower fitness values ranked first. A lower fitness value means that the simulated player or the scenario produces undesirable outcomes, which do not align with the intended objectives of the developers.

3.3.4 *CoEBA genetic operators*

To generate subsequent populations, genetic operators are applied, which involve selecting the individuals that will serve as parents for the new individuals. Following the selection operator, other genetic operators, such as crossover and mutation, are applied to manipulate the individuals. These operators help in generating new sets of individuals from the existing ones. The selection process ensures that the fittest individuals are more likely to be selected as parents, thus increasing the chances of producing offspring with better fitness scores. Furthermore, the use of genetic operators like crossover and mutation helps to introduce diversity into the population, preventing premature convergence and promoting exploration of the search space.

Selection. In order to select individuals from both subpopulations, our approach considers their fitness values. Our approach employs the roulette wheel selection mechanism. This selection approach helps to prevent premature convergence [1], a phenomenon that can reduce the algorithm's performance. In essence, the roulette wheel selection mechanism assigns a probability of selection to each individual in the population that is directly proportional to their fitness score. By using this technique, the algorithm guarantees that solutions with greater fitness have a higher chance of being selected, emulating the concept of "survival of the fittest". This aids in the convergence towards optimal or nearly optimal solutions. Meanwhile, the roulette wheel selection mechanism maintains diversity by providing less fit solutions with a possibility of being chosen. This diversity is crucial as it prevents the algorithm from becoming trapped in local optima and encourages the exploration of the solu-

tion space, which is especially critical for complex optimization problems. The selected individuals will be the ones that generate the next individuals.

Crossover. In the CoEBA approach, the crossover operator plays a significant role in generating new and diverse individuals. This operator involves the combination of genetic material from two parent solutions to create offspring solutions with new characteristics. CoEBA uses a single, random, cut-point crossover, where two-parent solutions are split at a random cut-point to generate two sub-vectors. The offspring solutions are then produced by combining the first part of one parent with the second part of the other parent for the first offspring and combining the first part of the second parent with the second part of the first parent for the second offspring. It should be noted that the length of the solutions remains constant during the crossover operation. Thus, the new solutions obtained through the crossover operation are the same length as the parent solutions. This method helps to promote diversity and can lead to the discovery of new individuals that may not have been present in the previous population.

Mutation. In our approach, the mutation operator plays a crucial role in maintaining diversity in the population and exploring new regions of the search space. The mutation operator randomly changes one or more parameters in a solution, introducing novel genetic material that can lead to improved solutions. For the simulated player population, the mutation operator starts by selecting certain positions within the vector representation of the individual. These selected dimensions are then replaced by a different value of the corresponding parameter. However, the values for mutation are not randomly generated numbers. Instead, they are obtained from a catalog of values compiled by developers based on their experience. Similarly, the mutation operator for the scenario population also starts by selecting certain positions within the vector representation of the individual. These selected dimensions are then replaced by a different value of the corresponding environmental element. By utilizing a pre-collected catalog of values, the mutation operator can efficiently and effectively explore the search space of both populations. The use of a pre-collected catalog of values also ensures that the mutations are relevant and appropriate for the game's mechanics, increasing the likelihood of finding effective and realistic solutions.

CoEBA generates a collection of simulation traces ranked according to their expected probability of triggering the intended bug. Overall, the aim of CoEBA is to highlight potential sources of bugs in the interaction between the software models of the video game.

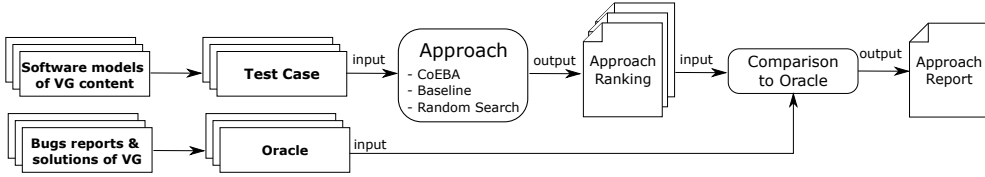


Figure 3.8: Overview of the evaluation process.

3.4 Evaluation methodology

This section presents the methodology used in the evaluation of the approach, including the preparation of the evaluation, the experimental process, the measures used to evaluate the algorithm performance, and the explanation of the statistical analysis performed.

In order to evaluate the CoEBA approach, we have formulated the following research questions:

- RQ_1 : What is the performance in terms of solution quality of the CoEBA, the baseline, and the random search approaches?
- RQ_2 : Is there any improvement in performance taking into account model interaction?
- RQ_3 : Are the differences in performance results obtained by the CoEBA, the baseline, and the random search approaches significant?

Answering the first research question (RQ_1) enables us to evaluate and contrast the performance outcomes of our proposed approach with the baseline approach, using metrics such as recall, precision, and F-measure. Additionally, we compare our approach and a random search (RS) as a sanity check. The second research question (RQ_2) seeks to determine whether model interactions affect the results. Finally, answering the third research question (RQ_3) allows us to conduct a thorough comparison between the approaches, provide formal and quantitative evidence (statistical significance) that the differences in the results are not due to chance, and demonstrate that these differences are significant in practice (i.e., effect size).

3.4.1 *Evaluation preparation*

The Kromaia developers provide the software models, the bug reports, the bug solutions, and the video game documentation. We employ all this information to build the oracle used in this evaluation. The oracle is the ground truth and is used to compare the results provided by the CoEBA approach, the baseline, and a random search.

The baseline is a previously developed approach for bug localization in GSE [14]. It uses an evolutionary algorithm for generating simulations that produce relevant traces for locating bugs. We use a standard random search (RS) as a sanity check. If RS outperforms an intelligent search method, we can conclude that there is no need to use a metaheuristic search. The algorithm starts with a random population of simulated players and a random population of scenarios. The simulated players are assessed using the CoEBA fitness function to obtain the best simulated player. Then, new random populations are generated and assessed. The search moves to a new simulated player if the fitness value is better than the current best simulated player. The loop is repeated until the stop condition is met.

To prepare the oracle, a total of 30 bugs were selected from the entire documentation. These 30 bugs cover the 6 different level bosses and have been provided by the Kromaia developers. These bugs are the most common when developers create the models.

We created the test case needed as input by the approaches for each bug. Each test case included the set of product models where the bug was manifested.

3.4.2 *Experimental process*

Figure 3.8 shows an overview of the process followed to evaluate our approach. The inputs to this process are used to run the different approaches, generating a set of results for each bug.

However, since these approaches perform genetic operations, chance could affect the results. Therefore, each approach is executed 30 times for each bug in order to minimize the effect of chance, as recommended in [7]. We employed the same parameters for the baseline approach as reported in [14]. For CoEBA, we began with the parameters reported in [11], since we utilized the same simulation algorithm and verified that they converged.

Once the approaches are executed, the next step is to compare them with each other and with a RS approach using statistical methods. To do this, the best solutions for each bug are selected from the ranking generated by each approach [23]. Then, these solutions are compared to the oracle (the actual solution) to generate the final report.

The approaches generate a list of traces. The trace contains all the model elements used by the interpreter at runtime during the simulation. All model elements that appear in the trace form the most relevant model fragment for the bug according to the trace. We can then compare the model fragments with an oracle to check accuracy.

Overall, the process involves running different approaches multiple times for each bug, selecting the best solutions, and comparing them with the actual solution to generate a report. This process is designed to minimize the effect of chance and provide a statistically rigorous evaluation of the different approaches.

3.4.3 Algorithm performance

A confusion matrix is a table that is commonly used to evaluate the performance of a classification model (our approach under evaluation) on a set of test data (the solution) where the truth values are known (derived from the oracle). In our case, each solution is a trace that consists of a subset of model elements that are part of a model (where the bug is present). Since the classification granularity is at the level of model elements, the presence or absence of each element is considered a classification. Thus, our confusion matrices distinguish between two values: TRUE (presence) and FALSE (absence). The confusion matrix then arranges the comparison results into four categories:

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

The confusion matrix holds the results of the comparison between the predicted and actual results; it is just a specific table layout to help the visualization of the performance of a classifier. However, to evaluate the performance of the approach, it is necessary to derive some measurements from the values of the confusion matrix. In this case, the three measurements are precision, recall, and F-measure.

- Precision measures the number of elements from the prediction (the result of the approach) that are correct according to the ground truth (the oracle).

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

- Recall measures the number of elements of the ground truth (the oracle) that are correctly retrieved by the prediction (the result of the approach).

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

- F-measure combines both recall and precision as the harmonic mean of precision and recall.

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3.3)$$

Precision and recall values can range between 0% to 100%. A value of 100% in precision and 100% in recall implies that both the predicted solution and the solution from the oracle are the same.

3.4.4 Implementation details

We conducted some parameter optimization to determine the optimal values for our algorithm's parameters. Nevertheless, this paper is not meant to optimize the algorithm's performance on a specific problem, but instead to compare the quality of different solutions provided by the algorithms. Hence, most of the default values suffice to evaluate the performance of search-based techniques within the testing context [8, 27]. We apply the crossover operation with a probability of 0.9 and the mutation operation with a probability that depends on the number of hulls in the boss during simulated player evolution or on the

number of environmental elements during scenario evolution: $1/(\text{Number})$. The population size is set to 100.

In addition, there are two fundamental performance metrics for evolutionary algorithms: one measuring solution quality and the other measuring algorithm speed. This study is focused on solution quality, which is measured through precision, recall, and F-measure (Section 3.4.3). Therefore, a budget has been assigned for each execution of the approach to ensure the quality of the solution. The focus of this paper is the solution quality and not the performance of each algorithm in terms of time. After conducting preliminary tests, we have determined that the time of convergence, or the point at which the search reaches its highest point and ceases to improve, occurs after around 40 seconds of execution. To ensure convergence, we set a wall clock time of 60 seconds for each run.

The replication package including models, bugs, oracles, and source code, is available at <https://doi.org/10.5281/zenodo.10231570>.

3.4.5 Statistical analysis

To compare our approaches accurately, we analyzed all the data derived from the empirical analysis using statistical methods in accordance with the guidelines provided in [7]. Our statistical analysis aims to: 1) provide formal and quantitative evidence (statistical significance) that our approaches do, in fact, have an impact on the comparison metrics (i.e., that the variations in the results were not achieved by mere chance); and 2) show that those differences are significant in practice (effect size).

Statistical significance

Hypotheses are used in statistical analysis to make inferences and draw conclusions about a population based on sample data. In the context of comparing algorithms, hypothesis testing helps determine whether there is sufficient evidence to support the idea that there is a difference between the algorithms being compared.

The null hypothesis, denoted as H_0 , represents the default assumption that there is no significant difference between the algorithms. It assumes that any observed differences in the data are due to random variation or chance. In other words, H_0 states that there is no effect or relationship between the algorithms being compared.

The alternative hypothesis, denoted as H_1 , is the opposite of the null hypothesis. It suggests that there is a significant difference between at least one pair of algorithms being compared. H_1 states that the observed differences in the data are not merely due to chance but are instead a result of a true difference or effect.

When conducting hypothesis testing, the goal is to gather enough evidence from the data to either reject the null hypothesis in favor of the alternative hypothesis or fail to reject the null hypothesis. This is done by calculating a test statistic and comparing it to a critical value or using a p – *value* to determine the statistical significance of the results. If the evidence is strong enough to reject the null hypothesis, it provides support for the alternative hypothesis and indicates that there is a significant difference between the algorithms.

The p – *values* obtained from the test is used to make this determination, and a p – *values* under 0.05 is considered statistically significant [7]. Non-parametric techniques like the Quade test are suitable for analyzing data that does not follow a normal distribution, and it is more powerful than other tests for real data [20] with a low number of algorithms (no more than four or five algorithms) [16].

However, the Quade test cannot determine which algorithm performs the best. To perform this comparison, we need to conduct a post hoc analysis that compares the performance of each algorithm against all other alternatives through pair-wise comparisons.

Effect size

When comparing different algorithms, more is needed to determine solely whether there are statistically significant differences between them. It is equally important to assess the practical significance or magnitude of any improvements observed. This can be achieved through the use of effect size measures. One commonly used is the *Cohen's d* value [15]. The *Cohen's d* value ranges from 0 to infinity, with higher values indicating a more substantial effect size. A value of 0.8 and above represents a high interaction effect, while a value between 0.5 and 0.8 signifies a medium interaction effect, and a value between 0.2 and 0.5 represents a small interaction effect. Values ranging from 0 to 0.2 indicate almost no interaction effect.

3.5 Results

This section presents the results obtained by the three approaches: 1) our approach, CoEBA; 2) the baseline approach; and 3) the RS approach; in evaluating bugs in Kromaia. The results are presented in Table 3.1, which shows the mean values and standard deviations for each measure obtained by each approach. The measures used to compare them are recall, precision, and F-measure.

Table 3.1: Mean values and standard deviation of the Precision, Recall, and F-measure values obtained by each approach.

	Precision $\pm (\sigma)$	Recall $\pm (\sigma)$	F-measure $\pm (\sigma)$
CoEBA	45.76 $\pm$ 28.70	84.83 $\pm$ 33.92	53.87 $\pm$ 24.71
Baseline	36.34 $\pm$ 28.47	68.95 $\pm$ 42.87	44.46 $\pm$ 29.03
Random Search	02.14 $\pm$ 10.06	08.07 $\pm$ 25.34	02.64 $\pm$ 09.64

The CoEBA and baseline approaches obtained better results than the RS approaches. Furthermore, the CoEBA approach outperformed the baseline approach and obtained the best results in all three measures (precision, recall, and F-measure) for the first research question (RQ_1). Specifically, the CoEBA approach achieved an average value of 45.76% in precision, 84.83% in recall, and 53.87% in F-measure.

The second research question (RQ_2) asks whether the interaction between models contributes to the improvement in bug location. The results show that CoEBA achieved an improvement of 9.42% in precision, 15.88% in recall, and 9.41% in F-measure compared to the baseline approach, which did not take into account the interaction between models. Therefore, answering to RQ_2 , the results indicate that take into account the interaction of software models pays off for bug localization in GSE.

3.5.1 Statistical analysis

The p – value obtained in the Quade test is $< 2.2 \times 10^{-16}$ for recall, precision, and F-measure (results are significantly different).

In the post hoc analysis, we perform statistical tests to determine if there are significant differences in the results between specific pairs of algorithms. Table 3.2 presents the results of Holm’s post hoc analysis, which includes the p – value and performance indicators for the algorithms. Most p – values

Table 3.2: Holm’s post hoc p – values for each pair of algorithms.

	Precision	Recall	F-measure
CoEBA vs Baseline	0.0088	$< 2.2 \times 10^{-16}$	0.57
CoEBA vs RS	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$
Baseline vs RS	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$	$< 2.2 \times 10^{-16}$

are below the significance threshold of 0.05, indicating significant differences in performance between the algorithms. We can see that precision and recall values are significant.

Table 3.3: Cohen’s d values of the Precision, Recall, and F-measure values obtained by each approach.

	Precision	Recall	F-measure
CoEBA vs Baseline	0.329345	0.410637	0.349083
CoEBA vs RS	2.028632	2.563807	2.731206
Baseline vs RS	1.602291	1.728865	1.933129

Furthermore, Table 3.3 presents the *Cohen’s d* values for recall, precision, and F-measure obtained by each approach. Regarding the *Cohen’s d* values for the CoEBA and the baseline approaches, the results show a small effect size in precision but highlight a more significant effect on recall.

Table 3.4: Classification of the precision and recall results. Obtained from [22].

Measure	Acceptable	Good	Excellent
Precision	20% – 29%	30% – 49%	$> 50\%$
Recall	60% – 69%	70% – 79%	$> 80\%$

Although the effect size according to *Cohen’s d* is small, we must take into account Hayes et al. classification [22]. Their work classifies performance for precision and recall as *Acceptable*, *Good*, or *Excellent*, as shown in Table 3.4. We started from a baseline approach with *Good* results in precision and through CoEBA approach we also obtained *Good* results in precision, but bordering on *Excellent*. On the other hand, the baseline approach obtained *Acceptable* results in recall while CoEBA approach achieved *Excellent* results.

The statistical analysis performed allows us to answer the research question (RQ_3) that asks if the performance results obtained by the CoEBA, the base-

line, and the random search approaches are significant. Although the differences in F-measure values are not significant when we compare the CoEBA and the baseline approaches, the differences in precision and recall values are significant.

3.6 Discussion

The video game industry has seen a significant increase in popularity, but video game research has not received enough attention yet. Recent research by Politowski et al. [30] has highlighted the differences in testing between classical software engineering (CSE) and game software engineering (GSE) and the need for further work in automating playtesting. They conclude that video game testing is still a long way to go. Our work contributes to this effort by addressing the crucial role of software model interaction in video game testing. The behavior of video games as a whole also includes the interaction between video game software models, and it is essential to explore this area to improve game testing. By investigating the co-evolution of simulated players and scenarios in bug location, our work helps to take another step towards improving video game testing and ultimately enhancing the quality of video games.

The evaluation presented in this work provides a detailed analysis of the values for three performance metrics obtained by applying our approach to locate the bugs of a commercial video game. The study conducted by Hayes et al. in [22] classified the results obtained into three categories: acceptable, good, and excellent. According to the Hayes et al. criteria, the CoEBA approach obtains good results for precision and excellent results for recall. It is worth mentioning that the precision and recall values obtained by CoEBA are significantly higher than the values obtained by the baseline. These suggest that CoEBA is a promising approach for bug localization in video game software models. It is important to note that this study represents only a first step in exploring the potential of CoEBA. Future research could investigate its application to more complex video game scenarios that involve additional tasks, such as finding objects or completing missions.

The recent work of Politowski et al. [30] includes a survey for video game developers. The main concerns highlighted in the paper relate to the high cost of building testing tools and the need for general testing tools that can work with a wide range of game types and game engines. Our approach does not share the concerns mentioned earlier. Our approach does not require develop-

ing simulation algorithms from scratch. Non-Playable Characters (NPCs) are commonly developed in video games as part of the regular development process. These NPCs serve different purposes, such as being enemies, cooperating with players, or inhabiting the virtual world to increase realism. NPCs can be utilized as the raw material for simulated players and scenarios, reducing the effort required to apply CoEBA. In addition, our approach locates bugs that arise from the interaction of software models and, hence, can be used in games that use software models. However, many game engines use models to capture different aspects of the game.

Additionally, some respondents expressed concerns about the potential for AI to replace game testers. When game developers perform playtesting, they have to test the complete video game to check several things, such as if it is fun or has problems like bugs or glitches. The solutions proposed by our approach are ordered by how far they are from what the developers consider to be the ideal experience (and, in their opinion, the most fun). However, our approach is intended to locate bugs rather than determine if the game is fun. Our approach can be an ally so that game testers can focus on tasks that are difficult for a machine to perform. This alliance could save time in video game developments avoiding delays in the deadlines and postponement of the launch date.

Finally, the approach we have developed considers the environmental elements present in the scenario and uses this information to apply a probability of damage dependent on the type of element. However, this is just the beginning of the possibilities that this approach presents. For example, future work involves exploring the interactions between more complex environmental elements such as mini-bosses. Additionally, this work serves as a starting point for other types of simulations where the objective is not only to defeat the boss but also to complete missions or find specific objects within the game environment. The potential for further exploration and expansion of this approach is vast, and it presents exciting opportunities for future research in the field of video game development and bug localization.

3.7 Threats to validity

In the following section, we will discuss potential threats to the validity of our work. To identify these threats, we will refer to the guidelines provided by De Oliveira et al. [26].

Conclusion validity threats.

- *Not accounting for random variation:* We addressed this threat by performing 30 independent runs for each algorithm on each bug.
- *Not using formal hypothesis and statistical tests:* We followed established guidelines [8] to employ standard statistical analysis and avoid this threat. To address the lack of good descriptive analysis, we utilized a variety of metrics, including precision, recall, and F-measure, to analyze the confusion matrix obtained from the experiments.

Internal validity threats.

- *Poor parameter settings threat:* We used standard values for the algorithms that have been tested in similar algorithms for bug location in video games [11, 14].
- *Lack of real problem instances:* For the evaluation, we utilized a commercial video game and obtained the necessary problem artifacts directly from the game developers and documentation. We selected the same bugs as the baseline algorithm to ensure a fair comparison.

Construct validity threats.

- *Lack of assessing the validity of cost measures:* To ensure the validity of our approach, we conducted a rigorous comparison with both the baseline and the sanity check. Additionally, our evaluation employed established metrics, such as precision, recall, and F-measure, which have been widely used in previous software engineering studies [8].
- *Lack of clarity on data collection:* To mitigate this potential threat, we utilized the SDML models of the contenders to conduct the simulation and relied on two primary indicators, the victory percentage, and health level, which were obtained from the developers and utilized to evaluate the approaches.

External validity threats.

- *Lack of a clear object selection strategy:* We tested the effectiveness of our approach in a real-world context by evaluating it on a commercial video game. To ensure the relevance of our results, we collected instances directly from the game’s development team and documentation.

3.8 Related Work

This section summarizes the research efforts from two areas that are related to this work: bug localization in MDE and bug localization in video games using simulations.

3.8.1 *Bug localization in MDE*

Several techniques are employed for bug localization in models beyond the video game domain. In software development, models can play various roles and provide domain information useful in bug localization. Hence, bugs in models cannot be ignored in MDE and our work focuses on this area. In [4], the authors use models at runtime to locate bugs resulting from dynamic reconfigurations. They employ search-based software engineering to generate a list of reconfiguration sequences that could potentially lead to the model at runtime containing the bug. In [5], their approach considers the domain information embedded in models and metamodels, which is not present in the source code. Additionally, Arcega et al. [6] evaluate the application of existing model-based bug localization approaches to mitigate the effects of starting localization in the wrong place. They also consider software engineers' ability to refine the results at different stages. By combining their approaches with manual refinement, they obtain the best results. Finally, Khorram et al. [24] propose a generic framework for coverage computation and fault localization of domain-specific models. They consider a test suite for an executable model and analyze the model's execution traces to extract its covered elements which compose the coverage matrix for the test suite. The applied fault localization techniques are capable of identifying the defects injected in the models based on coverage measurements.

Despite the significance of the aforementioned works in bug localization within MDE, none of them are tailored toward video game development. These approaches were not created considering the specific characteristics of video games, nor have they ever undergone evaluations in the context of video games. Moreover, as far as we know, no research has been conducted on the impact of model interactions as a source of bugs in video game development.

3.8.2 Bug localization in video games using simulations

Testing video games is commonly referred to as playtesting since it requires testers to play the game to analyze it. However, relying solely on human playtesters has limitations, and there is a need for new approaches to automate game testing. Research efforts that introduce alternative methods, such as simulations, are becoming more prevalent to address these challenges [30].

Several recent research works have employed agents to automate video game testing. For instance, Albaghajati and Ahmed [2] proposed a co-evolutionary genetic algorithm-based approach that uses agents to test and verify video games. The approach involves two agents, one of which generates faulty game states while the other uses colored Petri nets to analyze the game states and detect errors. Ariyurek et al. [10, 9] explored the impact of different Monte Carlo Tree Search (MCTS) modifications on the ability of agents to find bugs in games while operating within specific computational budgets. They introduced various changes to the MCTS algorithm and demonstrated that such modifications could enhance the agents' capability to find bugs. Furthermore, they also evaluated the human-like qualities of the agents, taking into account the computational resources available. Pfau et al. [29] employ agents to play adventure games as fast as possible (similar to a speedrun). Their approach can be utilized to execute various Visionaire Engine-based games and for assessing daily builds, hardware compatibility, performance tests, and quality assurance playthroughs in a generic manner. Shirzadehhajimahmood et al. [31] introduced an agent-based testing framework with a search algorithm and on-the-fly model construction. Their approach enables the agent to navigate through the 3D game world and overcome obstacles while solving the testing tasks. They also employ an Extended Finite State Machine (EFSM) model to capture the general properties of the game, which assists in the search process.

The approaches mentioned above also employ player simulation to test games and locate bugs, but they do not consider software models as a potential source of bugs. Since models are more and more used in video game development [34], ignoring them as a possible cause of bugs is not advisable.

Casamayor et al. [14] developed an evolutionary algorithm that uses model simulations to locate bugs in software models. They evolve game simulations that generate traces to identify the location of bugs. However, our approach differs from theirs as our work considers that the interactions of software models that occur during simulations can cause bugs to appear.

3.9 Conclusion

Bug localization in video games that use Model-Driven Engineering (MDE) requires specialized approaches due to the unique characteristics of the game software. The interaction between software models during gameplay can be a significant source of bugs, particularly in games that provide non-linear experiences with various possibilities for player behavior and scenarios.

To address this issue, this work aimed to evaluate how the interaction between software models affects bug localization in video games. The paper compared two approaches: CoEBA, which co-evolves simulated players (player behavior) and scenarios (environmental elements), and the baseline approach which does evolve simulations but does not take into account model interaction. The evaluation was performed on Kromaia case studies.

The results of this paper showed that CoEBA provided significantly better bug localization results than the baseline approach. The results of this paper show that model interaction through co-evolving simulated players and scenarios could be a promising direction for future research in the field of bug location for video games.

Bibliography

- [1] Michael Affenzeller et al. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications*. United Kingdom: CRC Press, 2009. ISBN: 978-1-58488-629-7 (cit. on p. 100).
- [2] Aghyad Albaghajati and Moataz Ahmed. “A co-evolutionary genetic algorithms approach to detect video game bugs”. In: *Journal of Systems and Software* 188 (2022), p. 111261. ISSN: 0164-1212. DOI: <https://doi.org/10.1016/j.jss.2022.111261> (cit. on p. 114).
- [3] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2010.05.004> (cit. on p. 86).
- [4] Lorena Arcega, Jaime Font, and Carlos Cetina. “Evolutionary Algorithm for Bug Localization in the Reconfigurations of Models at Runtime”. In: *Proceedings of the 21th ACM/IEEE International Conference on Model*

- Driven Engineering Languages and Systems*. MODELS '18. Copenhagen, Denmark: Association for Computing Machinery, 2018, 90–100. ISBN: 9781450349499. DOI: 10.1145/3239372.3239392 (cit. on p. 113).
- [5] Lorena Arcega et al. “An approach for bug localization in models using two levels: model and metamodel”. In: *Software and Systems Modeling* 18.6 (2019), pp. 3551–3576. DOI: 10.1007/s10270-019-00727-y (cit. on p. 113).
- [6] Lorena Arcega et al. “Bug Localization in Model-Based Systems in the Wild”. In: *ACM Trans. Softw. Eng. Methodol.* 31.1 (2021). ISSN: 1049-331X. DOI: 10.1145/3472616 (cit. on p. 113).
- [7] Andrea Arcuri and Lionel Briand. “A Hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering”. In: *Software Testing, Verification and Reliability* 24.3 (2014), pp. 219–250. DOI: <https://doi.org/10.1002/stvr.1486>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1486> (cit. on pp. 103, 106, 107).
- [8] Andrea Arcuri and Gordon Fraser. “Parameter tuning or default values? An empirical investigation in search-based software engineering”. In: *Empirical Software Engineering* 18 (2013), pp. 594–623 (cit. on pp. 105, 112).
- [9] Sinan Ariyurek, Aysu Betin-Can, and Elif Surer. “Automated Video Game Testing Using Synthetic and Humanlike Agents”. In: *IEEE Transactions on Games* 13.1 (2021), pp. 50–67. DOI: 10.1109/TG.2019.2947597 (cit. on p. 114).
- [10] Sinan Ariyurek, Aysu Betin-Can, and Elif Sürer. “Enhancing the Monte Carlo Tree Search Algorithm for Video Game Testing”. In: *IEEE Conference on Games, CoG 2020, Osaka, Japan, August 24-27, 2020*. IEEE, 2020, pp. 25–32. DOI: 10.1109/CoG47356.2020.9231670 (cit. on p. 114).
- [11] Daniel Blasco et al. “An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering”. In: *J. Syst. Softw.* 171 (2021), p. 110804. DOI: 10.1016/j.jss.2020.110804 (cit. on pp. 89, 92, 99, 103, 112).

- [12] Mohamed Boussaa et al. “Competitive Coevolutionary Code-Smells Detection”. In: *Search Based Software Engineering*. Ed. by Günther Ruhe and Yuanyuan Zhang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 50–65. ISBN: 978-3-642-39742-4 (cit. on p. 97).
- [13] Andrew Borg Cardona, Julian Togelius, and Mark J. Nelson. “Competitive coevolution in Ms. Pac-Man”. In: *2013 IEEE Congress on Evolutionary Computation*. 2013, pp. 1403–1410. DOI: 10.1109/CEC.2013.6557728 (cit. on p. 97).
- [14] Rodrigo Casamayor et al. “Bug Localization in Game Software Engineering: Evolving Simulations to Locate Bugs in Software Models of Video Games”. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. MODELS '22*. Montreal, Quebec, Canada: Association for Computing Machinery, 2022, 356–366. ISBN: 9781450394666. DOI: 10.1145/3550355.3552440 (cit. on pp. 86, 89, 93, 95, 99, 103, 112, 114).
- [15] Jacob Cohen. “Statistical power analysis”. In: *Current directions in psychological science* 1.3 (1992), pp. 98–101 (cit. on p. 107).
- [16] William Jay Conover. *Practical Nonparametric Statistics, 3rd Edition*. USA: Wiley, 1999 (cit. on p. 107).
- [17] Crytek. *CRYENGINE / The complete solution for next generation game development by Crytek*. <https://www.cryengine.com>. [Online; accessed 21-November-2021]. 2002 (cit. on p. 86).
- [18] Ehab Z. Elfeky et al. “A Systematic Review of Coevolution in Real-Time Strategy Games”. In: *IEEE Access* 9 (2021), pp. 136647–136665. DOI: 10.1109/ACCESS.2021.3115768 (cit. on p. 97).
- [19] Epic Games. *Unreal Engine: The most powerful real-time 3D creation tool*. <https://www.unrealengine.com>. [Online; accessed 21-November-2021]. 1998 (cit. on p. 86).
- [20] Salvador García et al. “Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power”. In: *Information Sciences* 180.10 (2010). Special Issue on Intelligent Distributed Information Sys-

- tems, pp. 2044–2064. ISSN: 0020-0255. DOI: <http://dx.doi.org/10.1016/j.ins.2009.12.010> (cit. on p. 107).
- [21] Mark Harman and Bryan F Jones. “Search-based software engineering”. In: *Information and software Technology* 43.14 (2001), pp. 833–839 (cit. on p. 89).
- [22] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods”. In: *IEEE Press* 32.1 (2006). ISSN: 0098-5589. DOI: 10.1109/TSE.2006.3 (cit. on pp. 109, 110).
- [23] Hisao Ishibuchi, Yusuke Nojima, and Tsutomu Doi. “Comparison between Single-Objective and Multi-Objective Genetic Algorithms: Performance Comparison and Performance Measures”. In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 1143–1150. DOI: 10.1109/CEC.2006.1688438 (cit. on p. 104).
- [24] Faezeh Khorram et al. “From Coverage Computation to Fault Localization: A Generic Framework for Domain-Specific Languages”. In: *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2022. Auckland, New Zealand: Association for Computing Machinery, 2022, 235–248. ISBN: 9781450399197. DOI: 10.1145/3567512.3567532 (cit. on p. 113).
- [25] L. Miguel Antonio and C. A. Coello Coello. “Coevolutionary Multiobjective Evolutionary Algorithms: Survey of the State-of-the-Art”. In: *IEEE Transactions on Evolutionary Computation* 22.6 (2018), pp. 851–865. DOI: 10.1109/TEVC.2017.2767023 (cit. on p. 97).
- [26] Márcio de Oliveira Barros and Arilo Claudio Dias Neto. *Threats to Validity in Search-based Software Engineering Empirical Studies*. Tech. rep. 0006/2011. 2011 (cit. on p. 111).
- [27] A. Panichella et al. “Parameterizing and Assembling IR-Based Solutions for SE Tasks Using Genetic Algorithms”. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. 2016, pp. 314–325. DOI: 10.1109/SANER.2016.97 (cit. on p. 105).

- [28] Luca Pascarella et al. “How Is Video Game Development Different from Software Development in Open Source”. In: May 2018. DOI: 10.1145/3196398.3196423 (cit. on p. 87).
- [29] Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. “Automated Game Testing with ICARUS: Intelligent Completion of Adventure Riddles via Unsupervised Solving”. In: CHI PLAY ’17 Extended Abstracts. Amsterdam, The Netherlands: Association for Computing Machinery, 2017. ISBN: 9781450351119. DOI: 10.1145/3130859.3131439 (cit. on p. 114).
- [30] Cristiano Politowski, Yann-Gaël Guéhéneuc, and Fabio Petrillo. “Towards Automated Video Game Testing: Still a Long Way to Go”. In: *Proceedings of the 6th International ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation*. GAS ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 37–43. ISBN: 9781450392938. DOI: 10.1145/3524494.3527627 (cit. on pp. 89, 96, 110, 114).
- [31] Samira Shirzadehhajimahmood et al. “An Online Agent-Based Search Approach in Automated Computer Game Testing with Model Construction”. In: *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*. A-TEST 2022. Singapore, Singapore: Association for Computing Machinery, 2022, 45–52. ISBN: 9781450394529. DOI: 10.1145/3548659.3561309 (cit. on p. 114).
- [32] Unity Technologies. *Unity Real-Time Development Platform / 3D, 2D VR & AR Engine*. <https://unity.com>. [Online; accessed 21-November-2021]. 2005 (cit. on p. 86).
- [33] W. Eric Wong et al. “A Survey on Software Fault Localization”. In: *IEEE Trans. Software Eng.* 42.8 (2016), pp. 707–740. DOI: 10.1109/TSE.2016.2521368 (cit. on p. 87).
- [34] Meng Zhu and Alf Inge Wang. “Model-Driven Game Development: A Literature Review”. In: *ACM Comput. Surv.* 52.6 (2019). ISSN: 0360-0300. DOI: 10.1145/3365000 (cit. on p. 86, 114).

Part III

Discussion

Discussion

This part of the thesis discusses the results of this work, connecting the research questions posed at the beginning of the work with the results of the research articles presented in the chapters of Part II. Additionally, the discussion introduces the research lines of the new works started as a continuation of this research.

GSE continues to evolve with the adoption of new paradigms and methodologies to improve software quality, productivity, and maintainability. With the adoption of MDD in video game development, it is necessary to address methods of bug localization in models. In order to achieve the objective of this thesis, we posed three research questions. Each of these questions motivates one of the papers in the publications of this compendium. In this section, we discuss how the results obtained in the work done help us to answer the research questions of this thesis.

Research Questions

RQ₁: *What is the current performance of SBSE approaches for bug localization in MDD?*

The first work (*SPE'24*) examines the performance of an SBSE approach, specifically an evolutionary algorithm, for locating model fragment location problems (such as bugs or features) in MDD environments. The algorithm operates on populations of software models and also considers the nature of model fragment location problems (MFLPs) where the algorithm is applied. The evaluation involves searching 1,895 MFLPs, characterized by five key measures, across two industrial case studies and testing 625 unique search parameter combinations.

The findings reveal that variations in population size, replacement percentage, and crossover rate can lead to performance shifts of approximately 30%. Among problem characteristics, search space size has the most significant impact. While search parameters primarily affect precision, MFLP characteristics play a greater role in recall.

These results underscore the need to carefully balance both factors for optimizing SBMDE techniques. Although general trends were observed, optimal configurations vary by context. Classifying problems based on MD-MFLP measures helps design effective evolutionary algorithms for bug localization and maintenance in MDD environments.

RQ₂: *How do MDD and CcD compare in terms of performance, productivity, and user satisfaction for bug localization tasks in video game development?*

The second work (*CAiSE'24*) reveals that the comparison between MDD and CcD in terms of performance, productivity, and user satisfaction for bug localization tasks in video game development reveals that it is influenced by the expertise of the developers involved.

In terms of user satisfaction, both students and professionals reported a preference for MDD when performing bug localization tasks. The structured nature of models, combined with their higher level of abstraction, provided a positive perception among participants. However, this satisfaction did not directly translate into higher performance and productivity. When evaluating students, no significant differences were observed between MDD and CcD in terms of performance and productivity. In contrast, professionals exhibited a clear advantage in performance and productivity when working with CcD. This discrepancy suggests that while MDD is perceived favorably, its practical effectiveness in bug localization tasks remains uncertain, particularly for experienced developers who are more proficient in bug localization in traditional source code.

Additionally, the findings indicate that professionals, who rely on established bug localization practices, struggled more with bug localization in MDD than students, who were more willing to use the metamodel for guidance. This suggests that existing bug localization methodologies may be suboptimal for models, emphasizing the need to develop specialized techniques for bug localization in MDD.

Given the increasing adoption of MDD in video game development, there is a pressing need to refine bug localization strategies for models. The assumption that models inherently improve bug localization is not supported by empirical evaluations.

RQ₃: *Does considering model interactions improve the performance of existing bug localization techniques in video game development?*

As evidenced in the experiment carried out to answer RQ₂, it is necessary to implement better approaches that allow bug localization in models in video games.

In order to improve existing approaches, in the third work (*IST'24*) it is proposed to take into account the interaction of models for bug localization. The behavior of a video game emerges from the interplay of various software models, and bugs may result not only from isolated models but also from their combined execution during gameplay. This necessitates a shift in bug localization strategies.

Our evaluation of a commercial video game demonstrates that incorporating software model interactions into bug localization leads to significant improvements. Our approach has achieved high precision and recall metrics when

compared to the baseline approach, which did not take into account model interactions.

Despite the promise of these approaches, concerns remain about the potential replacement of human testers by AI-driven simulations. However, automated bug localization should be seen as an ally rather than a substitute. Although AI can efficiently identify structural and logic-based bugs, human testers remain essential for evaluating subjective aspects such as game enjoyment and narrative coherence. Automation can streamline the debugging process, allowing testers to focus on higher-level qualitative assessments rather than repetitive bug identification tasks.

Future Lines of Research

As explained in the introduction to this thesis, there is still much to explore and improve in the area of GSE. Some lines on which work has already begun with this work as a starting point are as follows.

Model Interaction

Future research will focus on deepening the understanding of model interactions, not only to improve bug localization but also to improve other software maintenance tasks in video game development. Expanding the scope of model interaction could lead to more effective bug localization, optimization, and adaptation of game mechanics. This work serves as a foundation for broader applications, including simulations where objectives extend beyond bug localization to tasks such as mission completion, object retrieval, and dynamic game balancing. The potential for further exploration is vast, offering opportunities to refine maintenance processes and improve the overall quality and stability of video game software.

Experiments and Case Studies in GSE

Empirical Software Engineering studies have evolved with increasing maturity and rigor, yet their application in the GSE context remains underdeveloped. This line of research aims to analyze the use of the terms *case study* and *experiment* in GSE literature to assess their accuracy and consistency. Preliminary observations suggest that these terms may often be misapplied, potentially more so than in traditional software engineering. The study will quantify the extent of these misclassifications and identify specific patterns of misuse.

Furthermore, beyond merely highlighting inaccuracies, this work will provide concrete examples, propose recommendations, and offer insights to improve methodological clarity in future GSE research.

Understanding the Role of Models in Video Games

Future research will seek to better understand why software models (e.g., Blueprints in case of Unreal and Unity Visual Scripting in case of Unity) have been so successful in the context of video game development. In contrast, game development environments frequently allow models to operate in parallel with code, providing intuitive, high-level abstractions while preserving the flexibility and performance of low-level programming. This line of research will explore the hypothesis that the coexistence of models and source code as equally important development artifacts is a key factor in their success. In contrast to conventional MDD approaches that emphasize model-to-code transformations, game development often adopts a hybrid paradigm wherein models and source code are regarded as equally significant artifacts throughout the development lifecycle.

Part IV

Conclusions

Conclusions

This part presents the final conclusions of the thesis, summarising the objective, the study carried out, and the results of our work.

The video game industry has become a highly profitable sector with increasingly complex development processes, leading to the rise of GSE. The growing adoption of MDD in games highlights the need for specialized bug localization methods. Bugs often stem from intricate interactions between game mechanics, player behavior, and the environment, making effective bug localization essential to reducing development time, preventing delays, and ensuring higher-quality releases. This thesis focuses on developing efficient bug localization techniques tailored to software models in game development.

The research carried out in this thesis has led to the publication of three articles in reputable and specialized journals, achieving positive results.

- In the first article of this thesis [3] the impact of search-based software engineering for MDD maintenance tasks is explored, highlighting how different search parameter vectors and model fragment location problem measures influence performance. The study suggests that the SBMDE community should consider the impact of search parameters and problem nature when evaluating results.
- The second article of this thesis [2] compares MDD and CcD in terms of performance, productivity, and satisfaction, focusing on bug localization in a commercial video game. The study found that MDD increased satisfaction, but professionals performed better in CcD. The techniques used for bug localization in MDD showed suboptimal results, indicating a knowledge gap. With the growing use of MDD in video games, the study emphasizes the need for effective bug localization methods tailored to MDD.
- In the third article of this thesis [1] proposes an approach (CoEBA) for bug localization in video games using MDE, highlighting the importance of interaction between software models. Two approaches were compared: CoEBA, which co-evolves simulated players and scenarios, and a baseline approach that does not consider model interaction. The results showed that CoEBA provided better bug localization than the baseline approach, suggesting that model interaction could be a promising direction for future research in bug localization for video games.

The findings of this thesis reinforce the necessity of advancing bug localization techniques in MDD for video games. Current approaches focus primarily on source code, but this research underscores the importance of incorporating software models and their interactions into the bug localization process. The insights gained contribute to the GSE community by highlighting how

model-based approaches can improve bug localization efficiency and software quality. Ultimately, this research aims to complement human testers by providing enhanced tools and methodologies, facilitating faster and more accurate bug localization, and supporting the development of more reliable and quality video games.

Bibliography

- [1] Isis Roca et al. “Co-evolving scenarios and simulated players to locate bugs that arise from the interaction of software models of video games”. In: *Information and Software Technology* 169 (2024), p. 107412. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2024.107412> (cit. on p. 132).
- [2] Isis Roca et al. “Comparing MDD and CcD in the Bug Localization Context: An Empirical Evaluation in Video Games”. In: *Advanced Information Systems Engineering*. Ed. by Giancarlo Guizzardi et al. Cham: Springer Nature Switzerland, 2024, pp. 581–595. ISBN: 978-3-031-61057-8 (cit. on p. 132).
- [3] Isis Roca et al. “On the interaction between the search parameters and the nature of the search problems in search-based model-driven engineering”. In: *Software: Practice and Experience* 54.8 (2024), pp. 1312–1336. ISSN: 0038-0644. DOI: <https://doi.org/10.1002/spe.3320>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3320> (cit. on p. 132).

