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

Isis Roca[1,2], África Domingo[1], Óscar Pastor[2], Carlos Cetina[1], and Lorena Arcega[1]

[1] Universidad San Jorge. SVIT Research Group, Zaragoza, Spain
{iroca, adomingo, ccetina, larcega}@usj.es
[2] Universitat Politècnica de València. PROS Research Center, Valencia, Spain
opastor@dsic.upv.es

Abstract. 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 that has been released on PlayStation 4 and Steam. 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.

Keywords: Experiment · Model-Driven Development · Code-centric Development · Bug Localization · Game Software Engineering.

1 Introduction

The video game industry has evolved into a highly profitable segment [3]. The increasing popularity of video games has led to a rise in the complexity of creat-

* Partially supported by MINECO under the Project VARIATIVA (PID2021-28695OBI00) and by the Gobierno de Aragón (Spain) (Research Group S05_20D)

[3] T. B. R. Company, Gaming global market report 2023, https://www.thebusinessresearchcompany.com/report/gaming-global-market-report.

ing 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 has emerged known as Game Software Engineering (GSE). 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) [4]. 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, the increase in video game complexity is accompanied by an increase in the appearance of software bugs. 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 [2]. While MDD is acknowledged for its advantages in software development [15], particularly in terms of development effort and quality [11], 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 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 [9, 13]. Existing research has primarily compared MDD and CcD in software development scenarios [11].

Studies by Krogmann and Becker [11], Kapteijns et al. [9], and Mellegård and Staron [13] 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. [12] 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. [17] further underscored the advantages of MDD, demonstrating shorter development times and fewer challenges when utilizing code generation.

Panach et al. [16] and Domingo et al. [7] delved into the contextual factors influencing the benefits of MDD. Panach et al. affirm its consistent delivery of higher-quality results. Chueca et al. [5] 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.

3 Background

The case study that we used to assess this experiment focuses on the video game Kromaia [4]. 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. 1. The complete metamodel comprises more than 20 concepts, 20 relationships, and over 60 properties. SDML models are interpreted

[4] For more information about Kromaia, you can view the official trailer released by Playstation: https://youtu.be/EhsejJBp8Go

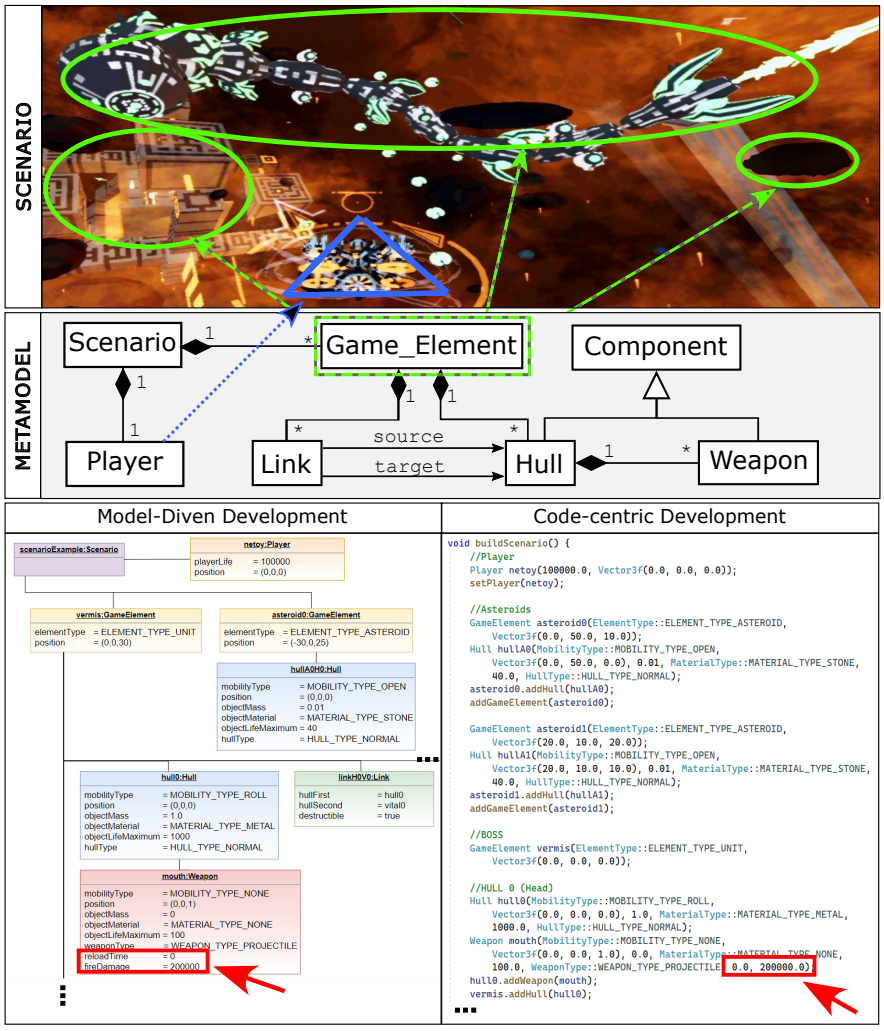

Fig. 1. Scenario Metamodel and Bug example in MDD and CcD.

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 [4]. For more information on the SDML model shown in Fig. 1, 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. 1. 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. 1. 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 total health, 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.

4 Experiment Design

4.1 Objectives

According to Wohlin's guidelines [20] for reporting software engineering experiments, we have organized our research objectives using the Goal Question Metric template for goal definition [3]. 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** inexperienced and experienced developers, **in the context of** bug localization for a video game company.

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.

To measure *Performance*, we built the confusion matrix of the subject's solution to the BL task to calculate its recall, precision, and F-measure. 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) [14]. 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.

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. [18]. 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.

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.

4.5 Participants

We selected the subjects using convenience sampling [20]. 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.

4.6 Experimental Objects

The tasks of our experiment were extracted from a real-world software development, Kromaia [4]. 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's 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 that was used for characterizing 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.

4.7 Experimental Procedure

The experiment was conducted in two different sessions. In the first session, the experiment was conducted face-to-face with the group of students (inexperienced developers). In the second session, the experiment was conducted online with professionals (experienced developers). 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 on it was available to the subjects during the experiment. (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.

4.8 Analysis Procedure

We have chosen the Linear Mixed Model (LMM) [19] 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) [18]. 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 dependent 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 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.

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

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 [10, 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 [6], 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 graphically describe the results.

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 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 1. Values for the mean and standard deviation of the response variables and Cohen d values for the alternatives of Method (MDD and CcD)

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

All of the statistical models shown in Formula 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 1 since it is the one that obtained the lowest values for the AIC and BIC fit measures. $Model\,1$ of Formula 1 is the one that best explains the changes for the variables related to satisfaction. Table 2 shows the results of the Type III fixed effects test for each

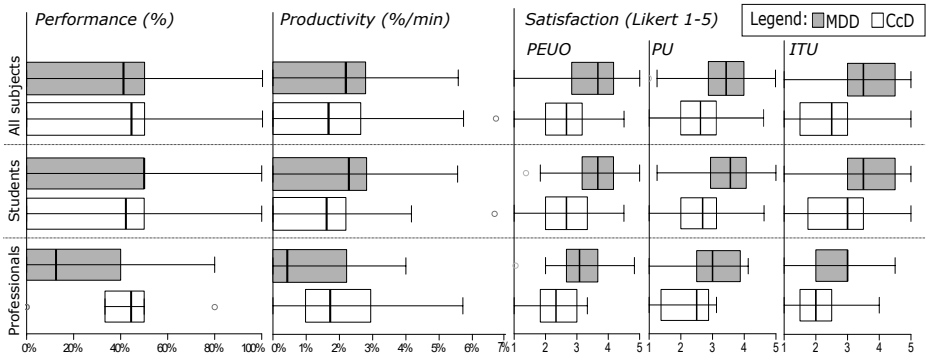

Fig. 2. Boxplots for dependent variables

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.

Table 2. Results of Type III test of fixed effects for each variable and factor. NA=Not Applicable

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

icant differences between the average results achieved by students and professionals on BL tasks. Nevertheless, as the **Method*Experience** 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.

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, preferred 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.

7 Threats to Validity

To describe the threats to validity of our work, we use the classification of Wohlin et al. [20]. 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 its involved a larger number of students than professionals, making inexperienced developers more represented in the overall results than experienced ones. 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.

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 comercial 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.

References

1. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: A systematic review. Information and Software Technology **52**(9), 888–901 (2010)

2. Arcega, L., Font, J., Haugen, Ø., Cetina, C.: An approach for bug localization in models using two levels: model and metamodel. Software and Systems Modeling **18**(6), 3551–3576 (2019)
3. Basili, V.R., Dieter Rombach, H.: The tame project: Towards improvement-oriented software environments. IEEE Transactions on Software Engineering (1988)
4. Blasco, D., Font, J., Zamorano, M., Cetina, C.: An evolutionary approach for generating software models: The case of kromaia in game software engineering. Journal of Systems and Software **171**, 110804 (2021)
5. Chueca, J., Trasobares, J.I., África Domingo, Arcega, L., Cetina, C., Font, J.: Comparing software product lines and clone and own for game software engineering under two paradigms: Model-driven development and code-driven development. Journal of Systems and Software **205**, 111824 (2023)
6. Cohen, J.: Statistical power for the social sciences. Hillsdale, NJ: Laurence Erlbaum and Associates (1988)
7. Domingo, Á., Echeverría, J., Pastor, Ó., Cetina, C.: Evaluating the benefits of model-driven development. In: Advanced Information Systems Engineering. pp. 353–367. Springer (2020)
8. Domingo, Á., Echeverría, J., Pastor, Ó., Cetina, C.: Comparing uml-based and dsl-based modeling from subjective and objective perspectives. In: Advanced Information Systems Engineering. pp. 483–498. Springer (2021)
9. Kapteijns, T., Jansen, S., Brinkkemper, S., Houet, H., Barendse, R.: A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. From code centric to model centric software engineering Practices Implications and ROI (2009)
10. Karac, E.I., Turhan, B., Juristo, N.: A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development. IEEE Transactions on Software Engineering (2019)
11. Krogmann, K., Becker, S.: A Case Study on Model-Driven and Conventional Software Development : The Palladio Editor. Software Engineering (2007)
12. Martínez, Y., Cachero, C., Meliá, S.: MDD vs. traditional software development: A practitioner's subjective perspective. In: Information and Soft. Technology (2013)
13. Mellegård, N., Staron, M.: Distribution of effort among software development artefacts: An initial case study. In: LN in Business Information Processing (2010)
14. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. ECIS 2003 proceedings p. 79 (2003)
15. Mussbacher, G., Amyot, D., Breu, R., Bruel, J.M., Cheng, B.H., Collet, P., Combemale, B., France, R.B., Heldal, R., Hill, J., et al.: The relevance of model-driven engineering thirty years from now. In: International Conference on Model Driven Engineering Languages and Systems. pp. 183–200. Springer (2014)
16. Panach, J.I., España, S., Dieste, Ó., Pastor, Ó., Juristo, N.: In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. Information and Software Technology (2015)
17. Papotti, P.E., Do Prado, A.F., de Souza, W.L., Cirilo, C.E., Pires, L.F.: A quantitative analysis of model -driven code generation through software experimentation. In: Advanced Information Systems Engineering. pp. 321–337. Springer (2013)
18. Vegas, S., Apa, C., Juristo, N.: Crossover designs in software engineering experiments: Benefits and perils. IEEE Transactions on Soft. Eng. **42**(2), 120–135 (2015)
19. West, B.T., Welch, K.B., Galecki, A.T.: Linear mixed models: a practical guide using statistical software. Chapman and Hall/CRC (2014)
20. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)