Evaluating the Benefits of Model-Driven Development

Empirical Evaluation Paper

África Domingo[1(✉)], Jorge Echeverría[1(✉)], Óscar Pastor[2(✉)], and Carlos Cetina[1(✉)]

[1] Universidad San Jorge, SVIT Research Group, Zaragoza, Spain
{adomingo,jecheverria,ccetina}@usj.es
[2] Universidad Politecnica de Valencia, PROS Research Center, Valencia, Spain
opastor@dsi.upv.es

Abstract. Researchers have been evaluating the benefits of Model-Driven Development (MDD) for more than a decade now. Although some works suggest that MDD decreases development time, other works limit MDD benefits to academic exercises and to developers without experience. To clarify the benefits of MDD, we present the results of our experiment, which compares MDD and Code-centric Development (CcD) in terms of correctness, efficiency, and satisfaction. Our experiment achieves fidelity to real-world settings because the tasks are taken from real-world video game development, and the subjects use domain frameworks as they are used in real-world developments. Our results challenge previous ideas that limit the benefits of MDD to academic exercises and to developers without experience. Furthermore, our results also suggest that understanding the benefits of MDD might require researchers to rethink their experiments to include the social part of software development.

Keywords: Model-Driven Development · Code-centric development · Empirical evaluation · Experiment

1 Introduction

Model-Driven Development (MDD) [19] promotes software models as the cornerstone of software development. In comparison to popular programming languages, these software models are less bound to the underlying implementation and are closer to the problem domain. Model transformation is at the heart of MDD since MDD aims to generate the software code from the models. This generation ranges from skeleton code to fully functional code systems.

For more than a decade, researchers have been evaluating the benefits of MDD [1,2,6–8,11–13,16–18]. Some works [6,11,18] conclude that MDD decreases development time (up to 89%) relative to Code-centric Development

Partially supported by MINECO under the Project ALPS (RTI2018-096411-B-I00).

S. Dustdar et al. (Eds.): CAiSE 2020, LNCS 12127, pp. 353–367, 2020.
https://doi.org/10.1007/978-3-030-49435-3_22

(CcD) [11]. Other works [8,17] suggest that gains might only be achieved in academic exercises. Furthermore, other works [8,12] assert that only developers without experience benefit from MDD. Therefore, more experimentation is needed to clarify the benefits of MDD.

In the context of MDD, domain frameworks (bodies of prewritten code) help model transformations to fill the abstraction gap between models and code (see Fig. 1 left). These frameworks are not exclusive of MDD. In the context of CcD, developers also use frameworks to accelerate development (see Fig. 1 right). However, previous experiments neglect the use of domain frameworks. This triggers the question of whether MDD benefits would hold when frameworks are considered.

In this work, we present our experiment, which compares MDD and CcD in terms of correctness, efficiency, and satisfaction. The tasks of our experiment are extracted from the real-world software development tasks of a commercial video game (Kromaia[1] released on PlayStation 4 and Steam). A total of 44 subjects (classified in two groups based on development experience) performed the tasks of the experiment. In our experiment, both MDD and CcD leverage a domain framework (see Fig. 1).

Our results challenge previous ideas that suggest that MDD gains are only achieved in academic settings. In our experiment with tasks from real-world development, MDD improved correctness and efficiency by 41% and 54% respectively. The results also contradict the previous claim that MDD benefits are limited to developers without experience. In the experiment, developers without experience (49% and 69%) as well as developers with experience (39% and 54%) significantly improved their results with MDD.

Furthermore, the results also uncover a paradox. Despite the significant improvements in correctness and efficiency, the intention of use of MDD did not achieve maximum values. We think this should influence future experiments to go beyond the technical facet and explore the cultural aspect of software development.

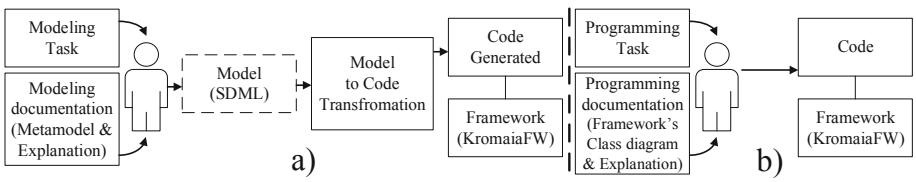

Fig. 1. a) Artifacts in the MDD task b) Artifacts in the CcD task

The rest of the paper is organized as follows: Sect. 2 reviews the related work. Section 3 describes our experiment, and Sect. 4 shows the results. Section 5 describes the threats to validity. Finally, Sect. 6 concludes the paper.

[1] https://youtu.be/EhsejJBp8Go.

2 Related Work

In the Motorola Case Study, Baker et al. [2] present their experiences deploying a top-down approach to MDD for more than 15 years in a large industrial context, resulting in a set of high-level strategies to enhance MDD deployment and success. Anda and Hansen [1] analyze the use of MDD with UML in software development companies. They show that developers who applied UML in modelling and enhancing legacy software experienced more challenges than those who modelled and developed new software from scratch. They propose a need for better methodological support in applying UML in legacy development.

Krogmann and Becker [11] present one of the first comparisons between MDD and CcD software development with respect to effort. Despite its limitations (it is not as a controlled experiment), their case study compares two software development projects, each of which was developed using a different method. They conclude that the model-driven approach could be carried out in only 11% of the time that of the code-centric approach. Another comparative case study of model-driven development with code-centric development by Kapteijns et al. [8] claims that MDD can be successfully applied to small-scale development projects under easy conditions. Heijstek and Chaudron [6] focused their case study on report specific metrics to analyze model size, complexity, quality, and effort. They analyze an industrial MDD project that was developed for the equivalent of 28 full-time team members. They showed that an increase in productivity, a reduction in complexity, and benefits from a consistent implementation are attributed to the use of MDD techniques. Mellegård and Staron [13] analyze the main modelling artefacts in the analysis and design phase of projects with respect to required effort and perceived importance. They conclude that the distribution of effort between models and other artefacts is similar in code-centric development to that of model-driven development. The factors for a successful adoption of MDD in a company were analyzed later by Hutchinson et al. [7] through interviews of 20 subjects from three different companies. They claim that a progressive and iterative approach, transparent organizational commitment and motivation, integration with existing organizational processes, and a clear business focus are required to guarantee success in the adoption of MDD techniques.

The experiment conducted by Martínez et al. [12] with undergraduate students compares three methods: Model-driven, Model-Based, and Code-Centric, regarding perceived usefulness, ease of use, intention of use, and compatibility. They conclude that the Model-Driven method is considered to be the most useful one, although it is also considered to be the least compatible with previous developers' experiences. Pappoti et al. [18] also present an experiment with a group of students in which an MDD based approach using code generation from models is compared with manual coding. When code generation was applied, the development time was consistently shorter than with manual coding. The participants also indicated that they had fewer difficulties when applying code generation. For Panach et al. [17], the benefits of developing software with MDD depend on the development context characteristics, such as problem complexity

and the developers' background experience with MDD. However, in a later work [16], where they report the results of six replications of the same experiment, they confirm that MDD yields better quality independently of problem complexity, and that the effect is bigger when the problems are more complex.

Table 1. Empirical studies on MDD

Work	Modelling Language	Domain Framework	Sample size	Context	Type of Study	Variables
[2]	UML	No	Not given	Industry	Case Study	Quality—Productivity
[1]	UML	No	28	Industry	Experiment	Difficulty—Use—Utility
[11]	DSL (GMFML)	No	11	Academia	Case Study	Quality—Efficiency Time effort
[6]	UML	No	4	Industry	Case Study	Quality—Effort Size—Complexity
[8]	UML	No	1	Industry	Case Study	Quality—Productivity Maintainability
[13]	UML	No	3	Industry	Case Study	Effort
[7]	UML	No	20	Industry	Case Study	Factors for a successful adoption of MDD
[12]	UML DSL (OOH4RIA)	No	26	Academia	Experiment	Perceived usefulness Perceived ease of use Intention to adopt Compatibility
[18]	UML	No	29	Academia	Experiment	Efficiency—Effort Participants' opinion
[17, 16]	UML	No	26	Academia	Experiment	Quality—Effort Satisfaction—Productivity
This work	DSL(SDML)	Yes	44	Academia[a]	Experiment	Correctness—Efficiency Satisfaction

[a] The tasks in our experiment are based on a real-world video game, but all participants involved are still students. Also, an experiment outside a company setting is by nature artificial.

Table 1 summarizes the related work. In contrast to previous works, we address the use of a domain framework as part of both MDD and CcD. This dimension has not been explored before. This contributes to achieving fidelity to real-world development since domain frameworks are fairly popular in both MDD and CcD contexts.

3 Experiment Design

3.1 Objectives

According to the guidelines for reporting software engineering experiments [22], we have organized our research objectives using the Goal Question Metric template for goal definition, which was originally presented by Basili and Rombach [3]. Our goal is to:

Analyze software development methods, **for the purpose of** comparison, **with respect to** correctness of the software developed, efficiency, and user satisfaction; **from the point of view of** novice and professional developers, **in the context of** developing software for a video game company.

3.2 Variables

In this study, the independent variable is the software development method (*Method*). It has two values, MDD and CcD, which are the methods used by subjects to solve the tasks.

Given that our experiment evaluates the benefits of MDD, and the most reported benefit of MDD is decreased development time, we consider two dependent variables, *Correctness* and *Efficiency*, which are related to the software that is developed. *Correctness* was measured using a correction template, which was applied to the programming artifacts developed by the participants after the experiment. *Correctness* was calculated as the percentage of passing assert statements with respect to the total number of assert statements. To calculate *Efficiency*, we measured the time employed by each subject to finish the task. *Efficiency* is the ratio of *Correctness* to time spent (in minutes) to perform a task.

We measured users satisfaction using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [14], wich is used for validating Information System Design Methods. We decompose satisfaction into three 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 intents to use a method. Each of these variables corresponds to specific items in the TAM questionnaire. We averaged the scores obtained for these items to obtain the value for each variable.

3.3 Design

Since the factor under investigation in this experiment is the software development method, we compared MDD and CcD. In order to improve experiment robustness regarding variation among subjects [21], we chose a repeated measurement using the largest possible sample size. To avoid the order effect, we chose a crossover design and we used two different tasks, T1 and T2. All of the subjects used the two development methods, each one of which was used in a different task.

The subjects had been randomly divided into two groups (G1 and G2). In the first part of the experiment, all of the subjects solved T1 with G1 using CcD and G2 using MDD. Afterwards, all of the subjects solved T2, G1 using MDD and G2 using CcD.

3.4 Participants

The subjects were selected according to convenience sampling [22]. A total of 44 subjects performed the experiment. There were 35 second-year undergraduate students from a technological program and 9 masters students in a subject about advanced software modelling currently employed as professional developers. The undergraduate students were novice developers and the master students were professional developers.

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 2 shows the mean and standard deviation of age, experience, hours per day developing software (Code Time) and hours per day working with models (Model Time). On average, all of the masters students had worked four years developing software. They worked on software development six hours per day while the undergraduate students, on average, dedicated less than 1.5 h to developing software each day. We used a Likert scale form 1 to 8 to measure the subjects' knowledge about domain-specific languages (DSL know) and programming languages (PL know). The mean and standard deviation of their answers are also in Table 2. All of them evaluated higher their programming language knowledge than their ability with models.

Table 2. Results of the demographic questionnaire

	Age $\pm\sigma$	Experience $\pm\sigma$	Code time $\pm\sigma$	Model time $\pm\sigma$	DSL Know $\pm\sigma$	PL Know $\pm\sigma$
Undergraduate	22.2 ± 0.4	0.6 ± 1.4	1.4 ± 0.8	1.0 ± 0.4	3 ± 1.7	4.2 ± 1.8
Masters	27 ± 2.6	4.3 ± 3.2	6.2 ± 2.0	0.9 ± 0.3	3.4 ± 1.0	6.2 ± 1.2
Total	21.6 ± 3.2	1.4 ± 2.4	2.4 ± 2.3	1 ± 0.4	3.1 ± 1.7	4.6 ± 1.9

The experiment was conducted by two instructors and one video game software engineer (the expert), who designed the tasks, prepared the correction template, and corrected the tasks. The expert provided information about both the domain-specific language and the domain framework. During the experiment, one of the instructors gave the instructions and managed the focus groups. The other instructor clarified doubts about the experiment and took notes during the focus group.

3.5 Research Questions and Hypotheses

We seek to answer the following three research questions:

RQ1. Does the method used for developing software impact the *Correctness* of code? The corresponding null hypotheses is H_{C0}: The software development method does not have an effect on *Correctness*.

RQ2. Does the method used for developing software impact the *Efficiency* of developers to develop software? The null hypotheses for *Efficiency* is H_{E0}: The software development method does not have an effect on *Efficiency*.

RQ3. Is the user satisfaction different when developers use different methods of software development? To answer this question we formulated three hypotheses based on the variables *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*:H_{PEOU}, H_{PU} and H_{ITU} respectively. The corresponding null hypotheses are:

H_{PEOU0}: The software development method does not have an effect on *Perceived Ease of Use*.
H_{PU0}: The software development method does not have an effect on *Perceived Usefulness*
H_{ITU0}: The software development method does not have an effect on *Intention to Use*.

The hypotheses are formulated as two-tailed hypotheses since not all of the empirical or theoretical studies support the same direction for the effect.

3.6 Experiment Procedure

The diagram in Fig. 2 shows the experiment procedure that can be summarises as follows:

1. The subjects received information about the experiment. An instructor explained the parts in the session, and he advised that it was not a test of their abilities.
2. A video game software engineer explained to the subjects the problem context and how to develop game characters on a video game with the domain framework to be used later in the experiment. The average time spent on this tutorial was 30 min.
3. The subjects completed a demographic questionnaire. One of the instructors distributed and collected the questionnaires, verifying that all of the fields had been answered and that the subject had signed the voluntary participation form in the experiment.
4. The subjects received clear instructions on where to find the statements for each task, how to submit their work, and how to complete the task sheet and the satisfaction questionnaire at the end of each task.
5. The subjects performed the first task. The subjects were randomly divided into two groups (G1 and G2) to perform the tasks with the domain framework. The subjects from G1 developed the first task coding with C++, and the subjects from G2 developed the task using MDD. The instructors used the distribution in the room to distinguish one group from another and to give specific guidance to each subject if requested.
6. The subjects completed a satisfaction questionnaire about the method used to perform the task.
7. The subjects answered an open-ended questionnaire about the method used to perform the task.
8. An instructor checked that each subject had filled in all of the fields on the task sheet and on the satisfaction questionnaire.

9. The subjects performed the second task exchanging methods. In other words, the subjects from G1 performed the second task using MDD, and the subjects from G2 performed the task using C++. Then, the subjects filled out the satisfaction questionnaire and the open-ended questionnaire that were related to the method used.
10. A focus group interview about the tasks was conducted by one instructor while the other instructor took notes.
11. Finally, the video game software engineer corrected the tasks and an instructor analyzed the results.

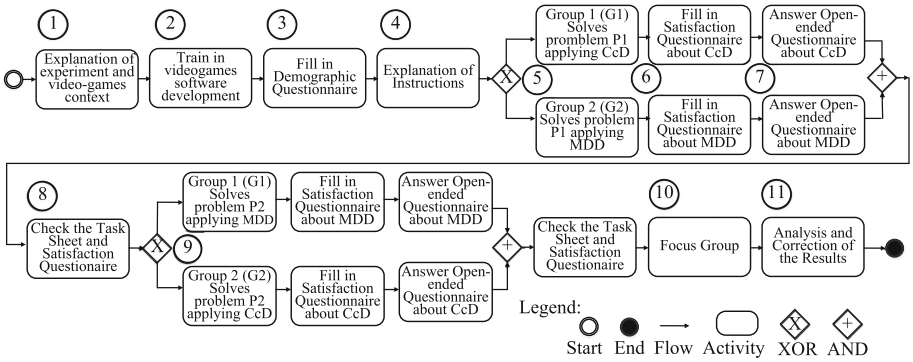

Fig. 2. Experimental procedure

In the tasks, the subjects were requested to develop the code of a part of the Kromaia video game, specifically a different game boss for each task. On average, the tasks took about 50 model elements of Shooter Definition Model Language (MDD), and about 300 lines of code (CcD). Learn more about developing the game bosses at https://youtu.be/Vp3Zt4qXkoY. The materials used in this experiment (the training material, the consent to process the data, the demographic questionnaire, the satisfaction questionnaire, the open-ended questionnaire, the task sheet and the materials used in the focus group) are available at http://svit.usj.es/MDD-experiment.

The experiment was conducted on two different days at *Universidad San Jorge* (Zaragoza, Spain) by the same instructors and a video game software engineer. On the first day, the experiment was performed by the masters students. On the second day, the undergraduate students performed the experiment. The masters and undergraduate students did not know each other. Their schedules at the university were completely different and the programs they followed are aimed at different types of student profiles.

4 Results

For the data analysis we have chosen the Linear Mixed Model (LMM) test [20] also used in others crossover experiments in software engineering [9]. LMM

handles correlated data resulting from repeated measurements. The dependent variables for this test are *Correctness*, *Efficiency*, *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*. In our study, the subjects are random factors and the *Method* used to develop software (MDD or CcD) is a fixed factor (the primary focus of our investigation). The statistical model (Model 1) used in this case is described as:

$$DV \sim Method + (1|\, subject) \tag{1}$$

Additionally, the subjects experience is also consider to be a fixed effect. We consider the variables *Experience* and the sequence *Method* and *Experience* to be fixed effects to account for their potential effects in determining the main effect of *Method* [9]. The statistical model (Model 2) used in this case is described in the following formula:

$$DV \sim Method + Experience + Method * Experience + (1|\, Subjec) \tag{2}$$

The statistical model fit for each variable has been 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 smaller AIC or BIC is considered to be the better fitting model. Additionally, the variance explained in the dependent variables by the statistical models is evaluated in terms of R^2 [9,15].

To quantify the difference between MDD and CcD, we have calculated the effect size using the means and standard deviations of the dependent variables of each method to obtain the standardized difference between the two means, Cohen's d Value [4]. 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. This value also allows us to measure the percentage of overlaps between the distributions of the variables for each method.

We have selected box plots and histograms to describe the data and the results.

4.1 Hypothesis Testing

The results of the Type III test of fixed effects for the fixed factors for each one of the statistical models used in the data analysis are shown in Table 3.

For all of the variables, *Method* obtained p-values less than 0.05, regardless of the statistical model used for its calculation. Therefore, all the null hypotheses are rejected. Thus, the answers to the research questions **RQ1**, **RQ2** and **RQ3** are affirmative. The method used for developing software has a significant impact on the correctness of code, efficiency, and the satisfaction of developers.

However, the fixed factors *Experience* and *Method*Experience* obtained p-values greater than 0.05, which implies that neither the developers experience nor the combination of both fixed factors had a significant influence on the changes in correctness of code, or the efficiency and the satisfaction of the developers.

Table 3. Results of test of fixed effects for each variable and each model

	Model 1	Model 2		
	Method	*Method*	*Experience*	*Method*Experience*
Correctness	(F = 643.3, p = .000)	(F = 137.7, p = .000)	(F = 2.5, p = .120)	(F = 1.9, p = .175)
Efficiency	(F = 1084.4, p = .000)	(F = 83.8, p = .000)	(F = 1.9, p = .171)	(F = .6, p = .447)
Ease of use	(F = 451.7, p = .000)	(F = 14.2, p = .001)	(F = .4, p = .508)	(F = .12, p = .730)
Usefulness	(F = 545.7, p = .000)	(F = 9.97, p = .003)	(F = 2.0, p = .168)	(F = 0.0, p = .965)
Intention to Use	(F = 341.4, p = .000)	(F = 5.3, p = .026)	(F = .2, p = .689)	(F = 1.1, p = .965)

4.2 Statistical Model Validity and Fit

The use of the Linear Mixed Model test assumed that residuals must be normally distributed. The normality of the errors had been verified by the Shapiro-Wilk test and visual inspections of the histogram and normal Q-Q plot. All of the residuals, except the ones carried out for *Efficiency*, obtained a p-value greater than 0.05 with the normality test. We obtained normally distributed residuals for *Efficiency* by using square root transformation. For the statistical analysis of the variable *Efficiency* with LMM, we used $DV = sqrt(Efficiency)$ in formulas (1) and (2). For the rest of the variables, DV is equal to their value.

Table 4. Comparison of alternative models for each variable

	Variance explained		Model fit			
	R^2		AIC		BIC	
	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
Correctness	63.8%	65.7%	−59.617	−56.313	−49.847	−46.638
Efficiency	57.9%	58.6%	−412.078	−398.820	−402.261	−398.097
Ease of use	4.8%	14.1%	232.593	232.281	241.917	244.004
Usefulness	10.9%	13.7%	218.295	217.452	228.113	227.176
Intention to use	4.8%	6.2%	270.298	268.847	280.115	278.570

The assessment of statistical model fit is summarized in Table 4. The values of the fit statistics R^2, AIC and BIC for each one of the statistical models are listed for each dependent variable. The fraction of total variance explained by both statistical models is similar. Model 2 obtained slightly better values of R^2, but the difference is not big. The AIC and BIC criteria were smaller for Model 1 in *Correctness* and *Efficiency*, and the difference was small for the other variables in favour of Model 2. This suggests that the *Method* factor explains much of the variance in the dependent variables. The factors incorporated in Model 2 with respect to Model 1 did not have a significant influence on the changes in the correctness of code, or the efficiency and satisfaction of the developers, as we have reported in Sect. 4.1.

4.3 Effect Size

The effect size of a Cohen d value of 2.63 for *Correctness* indicates that the magnitude of the difference is large. The mean of *Correctness* for MDD is 2.63 standard deviations bigger than the mean of *Correctness* for CcD. The mean for MDD is the 99.5 percentile of the mean for CcD. The box plots in Fig. 3(a) illustrate this result. This means that the distributions only have 9.7% of their areas on common, as shows Fig. 3(c).

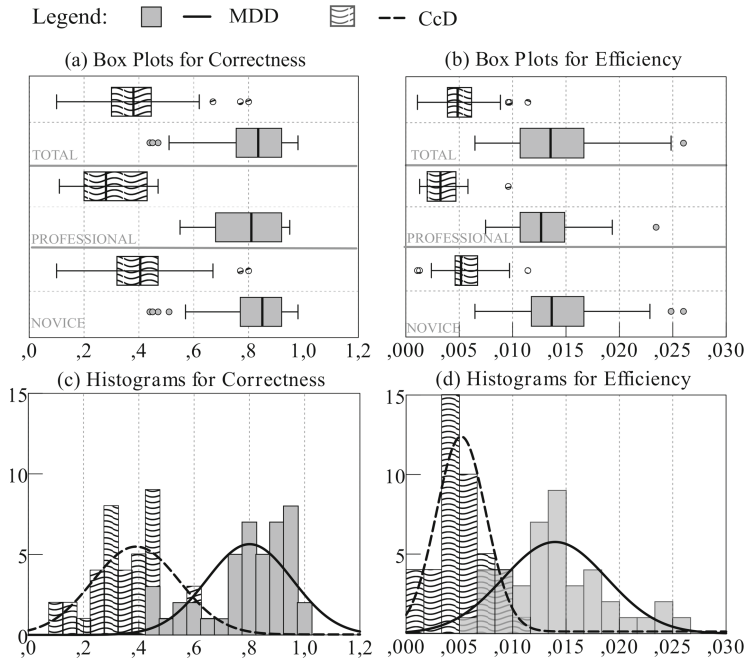

Fig. 3. Box plots and histograms for *Correctness* and *Efficiency*: (a) and (b) for *Correctness*; (c) and (d) for *Efficiency*

There is also a large effect size with a Cohen d value of 2.33 for *Efficiency*. The magnitude of the difference is also large. The mean for *Efficiency* for MDD, is 2.33 standard deviations bigger than the mean for efficiency for CcD. Again the mean for MDD is the 99.5 percentile of the mean for CcD (Fig. 3(b)) and the distributions of efficiency are different for 88% of their areas (Fig. 3(d)). The effect size of the differences based on using MDD or CcD for *Perceived Ease of Use* is medium-high, with a Cohen d value of 0.78. The box plots of Fig. 4(a) and the histograms of Fig. 4(d) illustrate how the differences in *Perceived Ease of Use* are not as great as the ones for *Correctness* or *Efficiency*. The magnitude of the difference between methods of development decreases to medium in the case of *Perceived Usefulness* with a Cohen d value of 0.69. Both the box plots of

Fig. 4(b) and the histograms of Fig. 4(e) illustrate a similar distribution to the one for *Perceived Ease of Use*. Again there is not a big difference in the diagrams corresponding to each subject group (professionals or novices)

Intention to Use obtained the lowest Cohen's d value (0.445), which means that the effect size of the method in this case is small to medium: the histograms of Fig. 4(f) shows that the distributions have much in common. The box plots of Fig. 4(c) shows that, in this case, the difference in the mean scores of *intention of use* (in favor of MDD versus CcD), is greater for the group of professionals than for the group of novices or the total group.

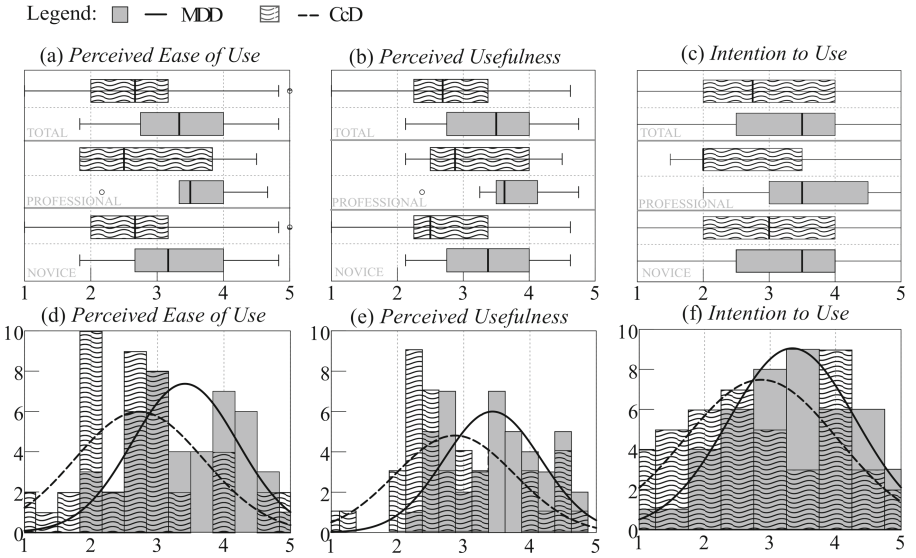

Fig. 4. Box plots and histograms for satisfaction: (a) and (d) for *Perceived Ease of Use*; (b) and (e) for *Perceived Ease of Use*; and (c) and (f) for *Intention to Use*

These data allow us to give more precise answers to **RQ1**, **RQ2**, and **RQ3**: for *Correctness* and *Efficiency*, the impact of the method used for development is very large, while, for satisfaction, the magnitude of the difference is medium.

4.4 Interpretation of the Results

Nowadays, domain frameworks are widely used in software development. The pre-implemented frameworks save developers time by using implementations that had been developed previously. In previous comparisons between MDD and CcD, one may argue that the benefits of MDD might come from the lack of a domain framework for CcD. This was not the case in our experiment.

Benefits must come from the software model itself and the model transformation (from model to code) that MDD adds to the framework. We still do not

know if the key lies on the abstraction of models, the automation of transformations, or a combination of these. However, it turns out that a framework on its own is not enough to achieve the benefits of MDD.

In our focus group interviews, both the professionals and the novices agreed that the abstraction of models is a double-edged sword. On the one hand, the subjects stated that models empower them to focus on the task at hand. On the other hand, the subjects stated that they lose control of the code generated. The subjects acknowledge that this loss of control negatively influences their intention of use. A few subjects stated that this loss of control would be alleviated if the model transformation were considered as another developer of the team. This triggers an interesting new direction of research, exploring the social implications of MDD on development teams.

5 Threats to Validity

To describe the threats of validity of our work, we use the classification of [22].

Conclusion Validity. The *low statistical power* was minimized because the confidence interval is 95%. To minimize the *fishing and the error rate* threat, the tasks and corrections were designed by a video game software engineer. Furthermore, this engineer corrected the tasks. The *Reliability of measures* threat was mitigated because the measurements were obtained from the digital artefacts generated by the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the treatment implementation was identical in the two sessions. Also, the tasks were designed with similar difficulty. Finally, the experiment was affected by the *random heterogeneity of subjects* threat. The heterogeneity of subjects allowed us to increase the number of subjects in the experiment.

Internal Validity. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience in software development. To mitigate this threat, the treatment was applied randomly. Other threat was *compensatory rivalry*: the subjects may have been motivated to perform the task with a higher level of quality by using the treatment that was more familiar to them.

Construct Validity. *Mono-method bias* occurs due to the use of a single type of measure [17]. All of the measurements were affected by this threat. To mitigate this threat for the correctness and efficiency measurements, an instructor checked that the subjects performed the tasks, and we mechanized these measurements as much as possible by means of correction templates. We mitigated the threat to satisfaction by using a widely applied model (TAM) [5]. The *hypothesis guessing* threat appears when the subject thinks about the objective and the results of the experiment. To mitigate this threat, we did not explain the research questions or the experiment design to the subjects. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To weaken this threat, at the beginning of the experiment the instructor explained to the subjects that the experiment was not a test about their abilities. *Author*

bias occurs when the people involved in the process of creating the experiment artifacts subjectively influence the results. In order to mitigate this threat, the tasks were balanced, i.e., their sizes and difficulty were the same for all treatments. Furthermore, the tasks were extracted from a commercial video game. Finally, the *mono-operation bias* threat occurs when the treatments depend on a single operationalization. The experiment was affected by this threat since we worked with a single treatment.

External Validity. The *interaction of selection and treatment* threat is an effect of having a subject that is not representative of the population that we want to generalize. However, using students as subjects instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts [10]. It would be necessary to replicate the experiment with different subject roles in order to mitigate this threat. The *domain* threat appears because the experiment has been conducted in a specific domain, i.e., video games development. We think that the generalizability of findings should be undertaken with caution. Other experiments in different domains should be performed to validate our findings.

6 Conclusion

In this work, we present an experiment that compares MDD and CcD in terms of correctness, efficiency, and satisfaction. Our experiment goes beyond the state of the art in terms of real-world fidelity and statistical power. A higher fidelity to real-world software development is achieved by means of the use of domain frameworks as they are used in real-world developments. Statistical power is enhanced by increasing the sample size. Our results challenge previous ideas that limit the benefits of MDD to academic exercises and to developers without experience. Furthermore, our results also suggest a new research direction that should include the social aspect of software development in order to better understand the benefits of MDD.

References

1. Anda, B., Hansen, K.: A case study on the application of uml in legacy development. In: ISESE 2006 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering (2006)
2. Baker, P., Loh, S., Weil, F.: Model-driven engineering in a large industrial context—motorola case study. In: Briand, L., Williams, C. (eds.) MODELS 2005. LNCS, vol. 3713, pp. 476–491. Springer, Heidelberg (2005). https://doi.org/10.1007/11557432_36
3. Basili, V.R., Rombach, H.D.: The tame project: towards improvement-oriented software environments. IEEE Trans. Softw. Eng. **14**(6), 758–773 (1988)
4. Cohen, J.: Statistical Power for the Social Sciences. Laurence Erlbaum and Associates, Hillsdale (1988)
5. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. **13**(3), 319–340 (1989)

6. Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: Conference Proceedings of the EUROMICRO (2009)
7. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: Proceedings - International Conference on Software Engineering (2011)
8. 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)
9. 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 Trans. Softw. Eng. (2019)
10. Kitchenham, B.A., et al.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. **28**(8), 721–734 (2002)
11. Krogmann, K., Becker, S.: A case study on model-driven and conventional software development : the palladio editor. In: Software Engineering (2007)
12. Martínez, Y., Cachero, C., Meliá, S.: MDD vs. traditional software development: a practitioner's subjective perspective. In: Information and Software Technology (2013)
13. Mellegård, N., Staron, M.: Distribution of effort among software development artefacts: an initial case study. In: Bider, I., et al. (eds.) BPMDS/EMMSAD -2010. LNBIP, vol. 50, pp. 234–246. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13051-9_20
14. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. In: ECIS 2003 Proceedings. p. 79 (2003)
15. Nakagawa, S., Schielzeth, H.: A general and simple method for obtaining r2 from generalized linear mixed-effects models. Meth. Ecol. Evol. **4**(2), 133–142 (2013)
16. Navarrete, J.I.P., et al.: Evaluating model-driven development claims with respect to quality: a family of experiments. IEEE Trans. Softw. Eng. (2018)
17. 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. Inf. Softw. Technol. **62**, 164–186 (2015)
18. 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: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 321–337. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38709-8_21
19. Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5), 19–25 (2003)
20. West, B.T., Welch, K.B., Galecki, A.T.: Linear Mixed Models: A Practical Guide Using Statistical Software. Chapman and Hall/CRC, Boca Raton (2014)
21. Wilde, N., Buckellew, M., Page, H., Rajilich, V., Pounds, L.T.: A comparison of methods for locating features in legacy software. J. Syst. Softw. **65**(2), 105–114 (2003)
22. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2