Deepening our Understanding on the use of Models and Code in Game Software Engineering: A Controlled Experiment in Unreal Engine

Jose Ignacio Trasobares*†, África Domingo*, Jorge Echeverría*, Lorena Arcega*, Carlos Cetina†
*SVIT Research Group, Universidad San Jorge, Zaragoza, Spain
{jtrasobares, adomingo, jecheverria, larcega}@usj.es
†VRAIN, Universitat Politècnica de València, Valencia, Spain
cetina@upv.es

***Abstract*—Video games are not just a work of art; they also involve a significant amount of programming. This programming part can be developed using code (C++) or software models (Blueprints), as is the case with the widespread Unreal game engine. In fact, in Unreal projects, it is common to use both code and software models. This work deepens our understanding of the use of code and software models in the software development of video games. To achieve this, we conducted a controlled experiment by comparing code and software models in Unreal. The experiment involved 31 active professional developers from 15 video game companies. Our results can help to explain the success of software models in video game workers.The results show that Blueprints benefit both developers and artists as effective development artifacts. Despite challenges in testing, modularization, and performance, they improve correctness. Their visual, constrained structure reduces errors and supports interdisciplinary collaboration.**

***Index Terms*—Model-Driven Development, Game Software Engineering, Empirical evaluation, Experiment.**

I. Introduction

The video game industry keeps growing in economic impact every year, and a significant portion of software developers contribute to this field. A report on the developer population indicates that 8.8 million out of 18.9 million active software developers have worked on video games at some point in their carers [1]. This prominence has led to the emergence of Game Software Engineering (GSE) [2]–[4], a subfield at the intersection of Software Engineering and video game development. GSE presents unique challenges that distinguish it from other domains, requiring a seamless integration of artistic and programming disciplines.

The development of video games requires a seamless fusion of artistry and programming expertise, resulting in development teams composed of diverse profiles. To facilitate this collaboration, modern game engines, such as Unreal Engine [5], incorporate features like Blueprint Visual Scripting [6], a node-based scripting system that enables game logic implementation without traditional coding.

The use of Blueprints in Game Software Engineering (GSE) aligns with Model-Driven Development (MDD) principles by abstracting complexity and allowing non-programmers to contribute to development. Although MDD has been extensively studied in classic software engineering, showing benefits, such as increased productivity, reduced complexity, and improved maintainability, it remains underexplored in Game Software Engineering (GSE).

This paper aims to bridge this knowledge gap by conducting an empirical study that compares models (Blueprints) and code (C++) in the context of the development of video games using Unreal Engine. Specifically, we evaluate their impact on correctness, efficiency, and developer satisfaction when performing game development tasks. Our study followed a crossover factorial design, comparing Blueprints and C++ across three tasks, with each participant performing different tasks with both methods. The study involved 31 active Unreal Engine professionals, including developers focused on areas such as Artificial Intelligence (AI), mechanics and User Interface (UI), as well as artists specializing in game design, animation and other artistic elements.

Our results confirm that both developers and artists benefit from using Blueprints. Participants reported that Blueprints are harder to test, merge, and have worse performance than code, yet Blueprints achieved significantly higher correctness. Satisfaction was similar between Blueprints and code. Statistical analysis showed no significant interaction between the method employed and the subject's role within a video game company, indicating that the advantages apply to both artists and developers. Focus group feedback emphasized that the constrained, visual nature of Blueprints helps prevent common programming errors.

The rest of this paper is structured as follows. Section II presents related work. Section III provides background information. Section IV details the experiment design, including objectives, research questions, and methodology. Section V presents the results of our empirical study, followed by a discussion in Section VI. Section VII outlines the threats to validity of our study. Finally, Section VIII concludes the paper.

II. Related Work

In the first part (Subsection II-A), we present the results of previous works that state the benefits and challenges of MDD in Classic Software Engineering. The second part (Subsection

II-B) presents the works on evaluating the adoption of MDD in Game Software Engineering.

A. MDD in Classic Software Engineering

The benefits of MDD in Classic Software Engineering have been analyzed through empirical studies for many years. Specifically, related work identifies benefits associated with MDD, such as reduced development effort and time (B1) [7]–[10]; increased productivity (B2), considering both software quality and development duration [11]–[13]; lower complexity and greater consistency in implementation (B3) [12]; improved developer satisfaction (B4) [13], [14]; enhanced software quality (B5) [9], [10], [15]; better team communication (B6) [16]; and cost reductions (B7) [9], [10].

Despite these advantages, MDD adoption in real-world projects remains limited [17]. Many organizations primarily use models for documentation rather than as integral parts of the development process, with most development efforts still relying on manual coding [18]. Several large-scale empirical studies involving over 5,000 participants [9], [10], [16], [17], [19]–[24] suggest that models are often used informally, a practice described as "modeling as sketch" [25]. Ultimately, code remains the central artifact, and software updates are typically implemented directly in code without synchronizing the underlying models.

The empirical studies also highlight key challenges in MDD adoption. Many researchers agree that inadequate UML training (C1) [19]–[21], [23], inadequate tools (C2) [9], [10], [16], [17], [21], [23], [24], [26] and lack of expertise (C3) [10], [21], [27] are major barriers. Additional challenges include low perceived usefulness (C4) [27], high investment costs (C5) [27], and the significant training effort required (C6) [9], [26].

B. MDD in Game Software Engineering

Trasobares et al. [28] studied the benefits of Software Product Lines (SPLs) compared to Clone and Own (CaO) in the context of video game models. A significant finding was the greater correctness of the elements developed with SPLs, although notable improvements in efficiency or satisfaction were not observed.

Subsequently, Chueca et al. [29] extended the previous work, by explicitly comparing the SPLs and CaO methods under the paradigms of both MDD and Code-Driven Development. Their results corroborated those of previous work, showing that the use of SPLs led to significantly higher correctness (over 23%) compared to CaO.

On the other hand, the research by Roca et al. [30] addressed the comparison between MDD and Code-Centric Development (CCD) from the specific perspective of bug localization in video games. Their findings revealed that while developers showed greater satisfaction when performing bug localization using MDD, the results in terms of performance and productivity varied depending on experience. For students, there were no significant differences, but professionals showed better performance and productivity when using CCD.

Martinez et al. [31] compare MDD with CCD for extended reality environments, using BOLT as an example of a model-based method. BOLT is a visual scripting tool in Unity game engine that resembles Unreal's Blueprints Visual Scripting. The work concludes that CCD provides detailed control over the application's logic and behavior. On the other hand, MDD offers a more intuitive and accessible approach.

Do Prado et al. [32] present an approach named Model-Driven Game Development (MDGD) that combines domain-specific languages (DSLs) with design patterns to provide flexibility and allow the generated code to be integrated with traditionally developed code. They conducted an experiment to validate the approach, finding that less experienced developers can create games faster and more easily, and the resulting product can be customized with manually written code. However, with the proposed approach, developers have less control over the code, making manual coding more difficult. Similarly, in [33] an MDD approach is presented for game development. The developer does not need to implement the game but rather model it. An experiment is conducted in which simplicity, extensibility, and attractiveness were positively evaluated. Furthermore, the results suggest that development time is reduced.

In conclusion, MDD has demonstrated significant benefits and challenges in Classic Software Engineering, its application in GSE remains largely unexplored: studies do not involve professorial video game developers as participants, Unreal (where models are the biggest success) is not studied by any work, no work aims to understand the reasons to use models in video games. This gap highlights the need for further investigation into MDD and video games.

III. Background

A game engine is a fundamental tool in video game development, that streamline the production of games. It includes functionalities, such as rendering graphics (2D and 3D), simulating physics, managing memory and assets, facilitating audio output, networking for multiplayer games, artificial intelligence for non player character behavior, and tools for animation and environmental effects. Many engines also support multi-platform development, making them a compelling choice for projects targeting multiple platforms by abstracting platform-specific complexities.

In fact, game engines have become the cornerstone of modern video game development, with 60% of developers using them as of 2022 [34]. Among the most widely adopted are Unity [35] and Unreal [5], although some studios opt to develop proprietary engines, such as CryEngine [36]. Unreal, in particular, has been a leading choice for game development since its launch in 1998 [34]. It has become the engine of choice for high-budget productions, powering many of the most highly anticipated titles today. In fact, nearly 80 of the most anticipated games released in 2023 are powered by Unreal Engine [37]. Beyond video games, Unreal has also expanded its influence in industries, such as film, television, architecture, automotive design, and simulation [38]. Furthermore, millions of players interact with games developed using

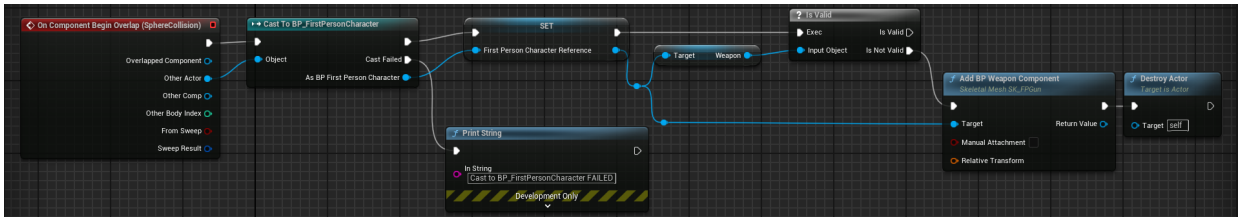


Fig. 1. Unreal Blueprint for a weapon pickup system.

Unreal, with Fortnite alone attracting more than 235 million monthly users [39].

One of Unreal's features is its Blueprint Visual Scripting system, a domain-specific modeling language that enables developers to create game logic using a node-based scripting environment rather than traditional code. Blueprints in Unreal are compiled into UnrealScript VM bytecode, which is executed on a virtual machine at runtime [6].

Fig. 1 presents a Blueprint script in Unreal that implements a weapon pick-up system and its destruction. The process begins when an actor overlaps with the *SphereCollision* component, which is configured to detect only the *Pawn* object type. Upon detecting an overlap event, the system attempts to cast the overlapping actor to *"BP_FirstPersonCharacter"*, ensuring that only the player character can interact with the pick-up. If the cast fails, a debug message is printed for development purposes. If the cast succeeds, the script checks whether the character already possesses a weapon using an *Is Valid* node. If the character does not have a weapon, the system attaches a custom *Skeletal Mesh Weapon* component (*"SkeletalMesh_SK_FPGun"*) to the character, integrating all necessary weapon logic and input handling. Once the weapon is successfully assigned, the pick-up actor (the instance of the weapon in the world) is destroyed to prevent multiple pick-ups of the same item.

Unreal also supports C++ development. Unlike Blueprints, which run on a virtual machine, C++ code is compiled directly into machine code, allowing for more complex and sophisticated game systems. This makes it the preferred choice for advanced features, such as physics simulations, and rendering optimizations. Additionally, C++ enables deeper customization of the engine, allowing developers to extend the core functionality of Unreal.

IV. Experiment Design

A. Objectives

To define objectives, our research follows the Goal Question Metric (GQM) model, initially introduced by Basili and Rombach [40]. Our goal is to **analyze** different specification methods, **for the purpose of** comparison, **with respect to** the correctness of the requested tasks, efficiency, and user satisfaction, **from the point of view of** experienced developers and artists, **in the context of** developing software for a using video game company.

B. Variables

In this study, the factor under investigation is the **Specification Method**, with two alternatives: using Blueprints (MDD) or using C++ code (CCD) to develop different types of tasks for a comercial video game.

The goal of this experiment is to evaluate the effects of different specification methods when developing different tasks using a specific game engine, Unreal. To assess this, we selected *Correctness* and *Efficiency* as the objective dependent variables, both of which are related to performance.

Correctness was measured using a correction template, which was applied to the tasks developed by participants after completing the experiment. The scores range from 0 to 100, representing the percentage of points achieved according to the correction template.

Efficiency was calculated by measuring the time each participant took to finish the task, based on the start and end times. *Efficiency* is defined as the ratio of *Correctness* to the time spent (in minutes) to complete the task.

Additionally, we examined the specification methods in terms of *Satisfaction*, using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [41]. We broke down *Satisfaction* into three subjective dependent variables as follows:

- **Perceived Ease of Use (PEOU)**: The degree to which a person believes that learning and specific specification method would require minimal effort.
- **Perceived Usefulness (PU)**: The degree to which a person believes that using a specific specification method would enhance performance.
- **Intention to Use (ITU)**: The degree to which a person intends to use a specific specification method.

Each of these variables corresponds to specific items in the TAM questionnaire. The scores for these items were averaged to obtain the value for each variable.

In addition to the main variables, the study will also take into account confounding factors that could alter the results due to the factor under investigation: the specification method. We will consider the effects of factors stemming from the design of the experiment, as well as the characteristics of the subjects participating in the study.

C. Design

We chose a factorial crossover design with three tasks (T1, T2, and T3), distributed across three periods. In the

first task, all subjects completed an initial task (T1) where they could freely choose between CCD and MDD. This task helped us capture their initial preference and their expertise. Performing this task allowed participants to become familiar with the experimental environment, the task format and the tools available, as well as helping them to identify and resolve configuration issues (e.g., software versions). In addition, this task helped us capture their initial preferences and their expertise.

In the second task (T2), the subjects were randomly divided into two groups (G1 and G2). G1 completed T2 using MDD (Blueprints), while G2 used CCD (C++) In the third task (T3), the groups switched methods: G1 completed T3 using CCD, while G2 used MDD. The factors that may affect the results in this crossover design are Period, with alternatives T2 and T3, and Sequence, with alternatives G1 and G2, and will be taken into account in the statistical analysis of the data.

This repeated measures design increases the sensitivity of the experiment [42]: observing the same subject using both alternatives controls between-subject differences, improving the experiment's robustness regarding variation among subjects. By using two different sequences for each group (G1 used Blueprints first and Code afterward, while G2 used Code first and Blueprints afterward) and different tasks, the design counterbalances some of the effects caused by using the alternatives in a specific order (e.g., learning effect, fatigue).

To verify the experiment design, we conducted a pilot study with one subject. This pilot study allowed us to estimate the time needed to complete tasks and questionnaires, detect typographical and semantic mistakes, and test the online environment used to conduct the experiment. The subject in the pilot study did not participate in the experiment.

D. Research questions and hypotheses

The research questions and null hypotheses are formulated as follows:

RQ1 - Does the **Specification Method** used in Unreal Engine impact the *Correctness* of software developed for video games? The corresponding null hypothesis is $H_{0,C}$: The **Specification Method** used in Unreal Engine does not have an effect on *Correctness*.

RQ2 - Does the **Specification Method** used in Unreal Engine influence the *Efficiency* of developers when creating software for video games? The null hypothesis for *Efficiency* is $H_{0,E}$: The **Specification Method** used in Unreal Engine does not have an effect on *Efficiency*.

RQ3 - Does user satisfaction differ when video game software is developed in Unreal Engine using different **Specification Methods**? 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:

- $H_{0,PEOU}$: The **Specification Method** used in Unreal Engine does not have an effect on *Perceived Ease of Use*.
- $H_{0,PU}$: The **Specification Method** used in Unreal Engine does not have an effect on *Perceived Usefulness*.
- $H_{0,ITU}$: The **Specification Method** used in Unreal Engine does not have an effect on *Intention to Use*.

E. Participants

The population of this study consists of working professionals in the video game industry who use Unreal in their routine tasks within a video game project. The study considers various primary profiles within the project team, namely developers, who are primarily oriented towards development (e.g. AI, mechanics, or UI developers), and artists, who are focused on artistic elements, game design or animation. To ensure the participation of professionals from the sector, we contacted video game studios that use Unreal in their developments, and invitations were also distributed through public mailing lists and specific social networks among different groups of game developers. Finally, 31 Unreal professionals agreed to participate in the study.

The subjects completed a demographic questionnaire that served to characterize the sample and to classify the subjects according to their role in the companies between artists (11 subjects) and developers (20 subjects) and their professional experience between juniors with less than two years of profesional experience (7 subjects) and seniors, with more than two years (23 subjects). Both characteristics will be included as blocking variables in the statistical analysis, since they could have an effect on the factor under investigation: the specification method.

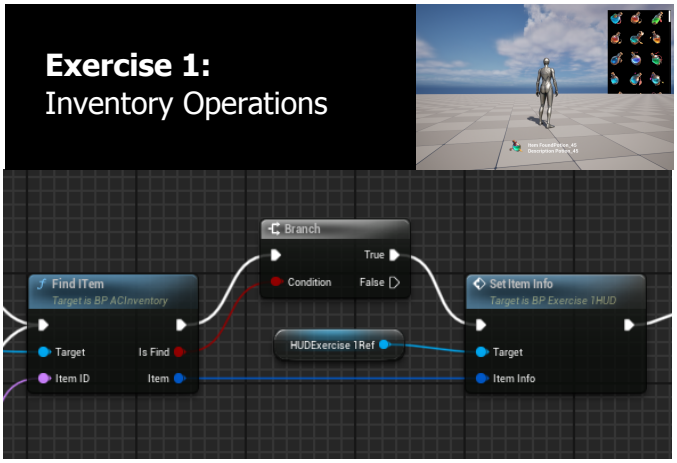
The experiment was carried out by three instructors. During the sessions, two instructors provided the necessary instructions, solved the doubts, and conducted the focus group. The three instructors took notes throughout the experiment and in the focus group.

F. Experimental objects

The tasks of our experiment were extracted from the work of Veron et al. [43], focusing on representative components commonly found in video games. Specifically, the tasks involve implementing a health system management component, an input processing component, and an inventory system component. These tasks were selected by experts to ensure a balanced difficulty level and relevance to real-world game development. Figures 2, 3, and 4 present a screenshot of each activity, along with its corresponding C++ code and Blueprint model, each incorporating one of the components under study.

The implementations were carried out in collaboration with professional video game developers using Unreal Engine 5 (version 5.3.2) and Microsoft Visual Studio Community 2022 (version 17.6.5). The implementations were reviewed by external professional developers to ensure consistency and correctness.

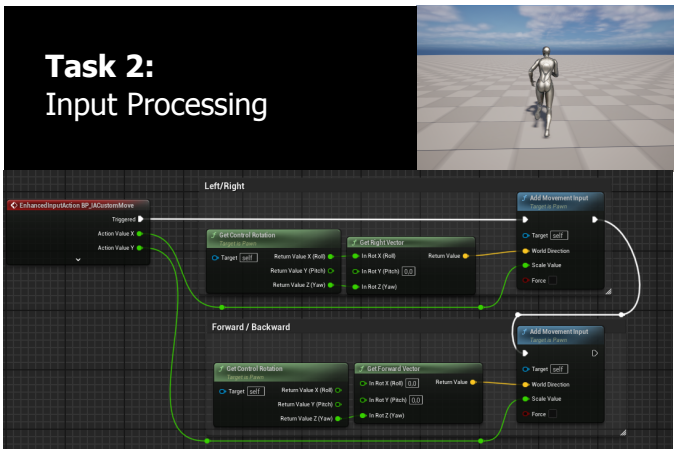
Fig. 2 shows, the inventory operations component involves an inventory with 157 items, where periodic searches are performed. In the Blueprint version, inventory items are instances of *BP_ItemData*, and searches are executed using the Unreal Blueprint node *Find*. In the C++ version, inventory items are stored in a struct named *FDataItem*, and searches




```
FDataItem* Item = Inventory.Find(ID);
if (Item) {
    ItemData = *Item;
    return true;
}
return false;
```

Fig. 2. Fragment of code (C++) and models (Blueprint) of task 1, where an object is searched for in the inventory.

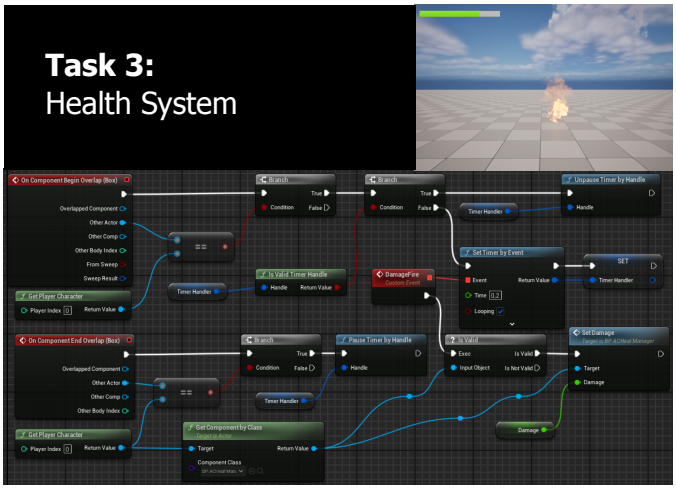
are performed within the class *UInventoryManager* using the *Find* function of the Unreal 5 class *TMap*.




```
FVector2D Axis2DValue = Value.Get<FVector2D>();
FVector Movement = FVector(Axis2DValue.X, Axis2DValue.Y, 0);
const FRotator Rotation = Controller->GetControlRotation();
const FRotator YawRotation(0, Rotation.Yaw, 0);
const FVector ForwardDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::X);
const FVector RightDirection = FRotationMatrix(YawRotation).GetUnitAxis(EAxis::Y);
AddMovementInput(ForwardDirection, Axis2DValue.Y);
AddMovementInput(RightDirection, Axis2DValue.X);
```

Fig. 3. Fragment of code (C++) and models (Blueprint) of task 2, where the input is processed for moving the character.

Fig. 3 shows the input processing component which enables character movement based on user inputs from a keyboard or controller. Both versions use Unreal's *Enhanced Input* system. The Blueprint version manages input mapping within *BP_CustomPlayer*, while the C++ version uses *AcustomPlayer* to bind inputs to movement functions.




```
void ADamageZone::OnOverlapBegin(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor,
                                 UPrimitiveComponent* OtherComp, int32 OtherBodyIndex,
                                 bool bFromSweep, const FHitResult& SweepResult) {
    ACharacter* PlayerCharacter = UGameplayStatics::GetPlayerCharacter(this, 0);
    if (OtherActor == PlayerCharacter) {
        StartDamageLoop();
    }
}

void ADamageZone::OnOverlapEnd(UPrimitiveComponent* OverlappedComponent, AActor* OtherActor,
                               UPrimitiveComponent* OtherComp, int32 OtherBodyIndex) {
    ACharacter* PlayerCharacter = UGameplayStatics::GetPlayerCharacter(this, 0);
    if (OtherActor == PlayerCharacter) {
        StopDamageLoop();
    }
}

void ADamageZone::StartDamageLoop() {
    ACharacter* PlayerCharacter = UGameplayStatics::GetPlayerCharacter(this, 0);
    LocalHealManager = Cast<UHealManager>(PlayerCharacter->GetComponentByClass(UHealManager::StaticClass()));
    GetWorldTimerManager().ClearTimer(DamageTimerHandle); // Clear any existing timer
    GetWorldTimerManager().SetTimer(DamageTimerHandle, this, &ADamageZone::DamageFire, 0.2f, true);
}

void ADamageZone::DamageFire(){
    if (LocalHealManager) {
        LocalHealManager->UpdateHealth(Damage);
    }
}

void ADamageZone::StopDamageLoop() {
    if (DamageTimerHandle.IsValid()) {
        GetWorldTimerManager().ClearTimer(DamageTimerHandle);
        DamageTimerHandle.Invalidate(); // Invalidate the handle
    }
}
```

Fig. 4. Fragment of code (C++) and models (Blueprint) of task 3, where the character receives a decrease in current health when damaged.

Fig. 4 shows the health system management component which involves a character entering into a damaging zone where its health decreases over time. In the Blueprint version, health variables are managed within a class named *AC_HealManager*, and damage is applied via *DamageZone*. The C++ version follows the same logic using the classes *UHealManager* and *ADamageZone*.

For data collection, we developed a questionnaire using Microsoft Forms with the following sections:

I 'Patient Information Sheet' that subjects must review and voluntarily accept. It clearly explains what the experiment consists of and how personal data will be handled.

II Demographic questionnaire to characterize the sample.

III Specific questionnaire to collect the observations of the subjects and their satisfaction with the tool and each of the methods used in the experiment. This questionnaire will be completed three times.

IV Final questionnaire to collect the the final observations

that the subjects may wish to provide.

The experimental objects used in this experiment (the tasks, the correction templates and the forms used for the questionnaires), as well as the results, the demographic analysis and the analysis of the subjects' opinions in the focus group, or the statistical analysis, are available as a replication package at https://doi.org/10.5281/zenodo.15133503

G. Experimental procedure

The experiment was conducted in three different on-line sessions. In each session, 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 two hours and in each session was conducted following the same experimental procedure outlined in Figure 5 and described as follows:

1) The experiment began with an explanation of the study, where subjects were introduced to its objectives, structure, and expectations. An instructor explained the different parts of the session and clarified that the experiment was not a test of their abilities. Subjects received detailed instructions on how the study would proceed, including the sequence of tasks, the tools they would be using, and the type of evaluations they would need to complete.
2) Then, the subjects were randomly divided into two groups, Group 1 (G1) and Group 2 (G2); the subjects from G1 received the links to access one form and the subjects from G2 received a link to another form. This questionnaire was designed to collect background information on each subject, including their role in their company or the level of experience in software development, familiarity with C++, and previous exposure to Blueprints.
3) After completing the demographic questionnaire, subjects proceeded to solve Task 1 (T1). At this stage, both G1 and G2 were allowed to freely choose whether to solve the task using Blueprints or C++. After completing the task, they reported which method they used. Following the task, subjects filled out a satisfaction questionnaire to assess their experience with Unreal Engine. During the performance of the first task, subjects can become familiar with the environment in which the experiment takes place and the type of tasks, and resolve any doubts and problems that may arise.
4) The subjects solve Task 2 (T2) using a specific method. The subjects from G1 had to use Blueprint to perform the task, and the subjects from G2 had to perform the same task but using C++. After submitting their solution, the subjects completed a satisfaction questionnaire about the method used for solving the task.
5) The subjects solve Task 3 (T3) using a different method. The subjects from G1 used C++, and the subjects from G2 used Blueprints. Then, the subjects completed the satisfaction questionnaire.
6) Finally, once all tasks were completed, all subjects took part in a focus group discussion. This session allowed them to openly discuss their experiences with Blueprints and C++, covering aspects, such as ease of use, efficiency, code clarity, and the applicability of each method in real-world development.

H. Analysis Procedure

We have chosen the Linear Mixed Model (LMM) [44] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows for the evaluation of factors influencing a crossover design, such as period, sequence, or subject, as well as other blocking variables like the professional experience or the role in a videogame company [42]. 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, the **Specification Method** was set as a fixed-repeated factor to compare the use of Blueprints (MDD) and C++ (CCD), with subjects treated as a random factor $(1|Subj.)$ to reflect the repeated measures design. The dependent variables (DV) for the analysis were *Correctness* and *Efficiency*, and three additional variables related to *Satisfaction*: *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to Use* (ITU).

To account for the potential effects of factors present in a crossover design on the main effect of **Specification Method**, we treated **Period**, with alternatives Task 2 and Task 3, and **Sequence**, with alternatives G1 and G2, as fixed effects. In addition, the folowing variables were treated as fixed factors in the statistical analysis: **Session**, with alternatives 1, 2 and 3, representing each of the different sessions in which the experiment was conducted, and the blocking variables **Role** and **Experience**, derived from the differences between the profiles of the subjects. **Role** has alternatives 'Developer' and 'Artist', and the experience variable had two alternatives: 'Junior', subjects with a professional experience of less than two years and 'Senior' subjects with professional experience of more than two years.

We evaluated different statistical models to determine which factors, apart from **Specification Method**, could best explain the variations in the dependent variables. These models are described mathematically in equation 1. The initial model (Model 0) focuses on the primary factor of this study, **Specification Method** (SM). Additional models (e.g., Models 1, 2, and 3) were also tested, incorporating other fixed factors hat could influence the dependent variables (DV).

$$\begin{array}{ll} (Model\ 0) & DV \sim SM + (1|\,Subj.) \\ (Model\ 1) & DV \sim SM + Period + Sequence + (1|\,Subj.) \\ (Model\ 2) & DV \sim SM + Role + SM * Role + (1|\,Subj.) \\ (Model\ 3) & DV \sim SM + Role + Experience\,(1|\,Subj.) \end{array} \tag{1}$$

We assessed the goodness-of-fit for the models using criteria, such as Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The model with the smallest AIC or BIC value was considered the best fit [28],

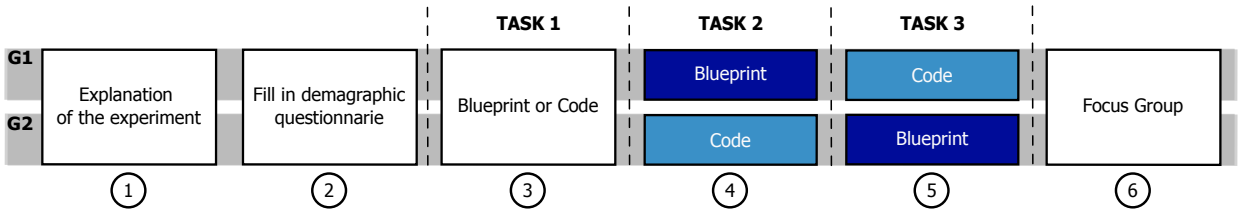


Fig. 5. Experimental procedure flowchart.

[29], [45], [46]. One of the assumptions for applying LMM is the normality of the residuals of the dependent variables. Only models that met this assumption were considered in the analysis. Normality was verified using Kolmogorov-Smirnov tests and visual inspection of histograms and Q-Q plots. To describe changes in the dependent variables, we selected the statistical model that met the residual normality assumption and also had the lowest AIC or BIC.

To quantify the differences in the dependent variables due to the factors considered, we calculated the Cohen *d* value [47], 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.

V. Results

There were differences in the means and standard deviations of all of the dependent variables depending on which **Specification Method** was used to solve the tasks in favor of MDD. However, while the differences in *Correctness* or *Efficiency* was large, the differences in the variables related to *Satisfaction* were small or medium. Table I shows the values for the mean and standard deviation of all the dependent variables considered for each one of the methods compared: MDD and CCD.

In addition, Table I shows the same descriptive statistics for the alternatives of the blocking variables related to the profile of the subjects: **Role**, with two alternatives (Developers and Artists), and **Experience**, also with two alternatives (Junior and Senior). Table I also shows the values of means and standard deviations by combination of the factor **Specification Method** with these blocking variables. This information allows us to analyze the effects that these blocking variables have on the dependent variables, as well as the effects that they may have on the differences between performing the tasks with one methodor another. In Table I the pairs of values are shaded according to the effect size of their differences. The darker the shade, the larger the difference in the values of the dependent variables across the alternatives of the factors and blocking variables considered. In addition, the italicized text highlights statistically significant comparisons.

The 25 pairs of box plots arranged in rows and columns in Fig. 6 illustrate the differences in *Correctness*, *Productivity*, and *Satisfaction* (*PEOU*, *PU*, and *ITU*) due to the blocking

TABLE I
Values for the mean and standard deviation of the response variables alternatives of Specification Method, Role, and Experience.

	Correctness	*Efficiency*	*Satisfaction* $\mu \pm \sigma$		
	$\mu \pm \sigma$	$\mu \pm \sigma$	*PEOU*	*PU*	*ITU*
All Subjects	37.24 ± 36.24	3.23 ± 5.29	3.14 ± 1.39	3.29 ± 1.14	3.47 ± 1.10
MDD	*49.59 ± 38.94*	4.98 ± 6.80	3.38 ± 1.41	3.52 ± 1.12	3.56 ± 1.13
CCD	*24.89 ± 28.97*	1.49 ± 2.09	2.89 ± 1.34	3.06 ± 1.13	3.37 ± 1.08
Developers	*51.47 ± 33.82*	*4.72 ± 6.06*	*3.33± 1.39*	3.41 ± 1.18	3.59 ± 1.14
MDD	65.79 ± 32.81	7.19 ± 7.57	3.68 ± 1.40	3.71 ± 1.17	3.73 ± 1.20
CCD	37.16 ± 28.98	2.24 ± 2.27	2.98 ± 1.31	3.12 ± 1.15	3.46 ± 1.09
Artists	*11.36 ± 24.57*	*0.54 ± 1.23*	*2.78 ± 1.34*	3.06 ± 1.04	3.24 ± 1.01
MDD	20.13 ± 32.00	0.97 ± 1.62	2.83 ± 1.32	3.17 ± 0.99	3.27 ± 0.98
CCD	2.60 ± 8.61	0.11 ± 0.36	2.73 ± 1.42	2.73 ± 1.42	3.20 ± 1.09
Junior	41.78 ± 37.53	3.36 ± 3.32	3.62 ± 1.44	3.59 ± 1.14	3.72 ± 1.18
MDD	60.82 ± 43.09	5.32 ± 3.70	3.93 ± 1.64	3.73 ± 1.43	3.65 ± 1.47
CCD	22.73 ± 18.71	1.40 ± 1.11	3.31 ± 1.25	3.45 ± 0.85	3.79 ± 0.91
Senior	35.92 ± 36.15	3.20 ± 5.77	3.00 ± 1.35	3.20 ± 1.14	3.39 ± 1.08
MDD	46.31 ± 38.00	4.88 ± 7.53	3.22 ± 1.33	3.45 ± 1.05	3.54 ± 1.05
CCD	25.52 ± 31.64	1.51 ± 2.32	2.77 ± 1.36	2.77 ± 1.36	3.24 ± 1.11

variables considered in the study. The first row of pairs of box plots corresponds to all of the subjects, and illustrates the differences in the dependent variables due to **Specification Method**. The following rows corresponds to the alternatives of the blocking variables considered in each dependent variable, and illustrates the differences due to this variable and its combination with **Specification Method**. The pair of box plots in the first row of Fig. 6 illustrate the effects of the factors **Specification Method** and the blocking variables **Role** and **Experience** in the dependent variables. The blocks of boxplots by fixed factor of Fig. 6, after the first row of boxplots, show the absence of differences of the blocking variables combined with method, since the differences between MDD and CCD do not depend on the alternatives of the factors **Role** or **Experience**. For all the dependent variables and in all alternatives of these factors, MDD obtain better results than CCD. However, for the variables related to *Satisfaction* the differences between methods are smaller than the differences in *Correctness* or *Efficiency*.

The statistical linear mixed model used to explain the statistical significance of the changes in the response variables includes all the fixed factors and the combination of **Specification Method** with the rest of fixed factors. The statistical model can be expressed mathematically as $DV \sim$

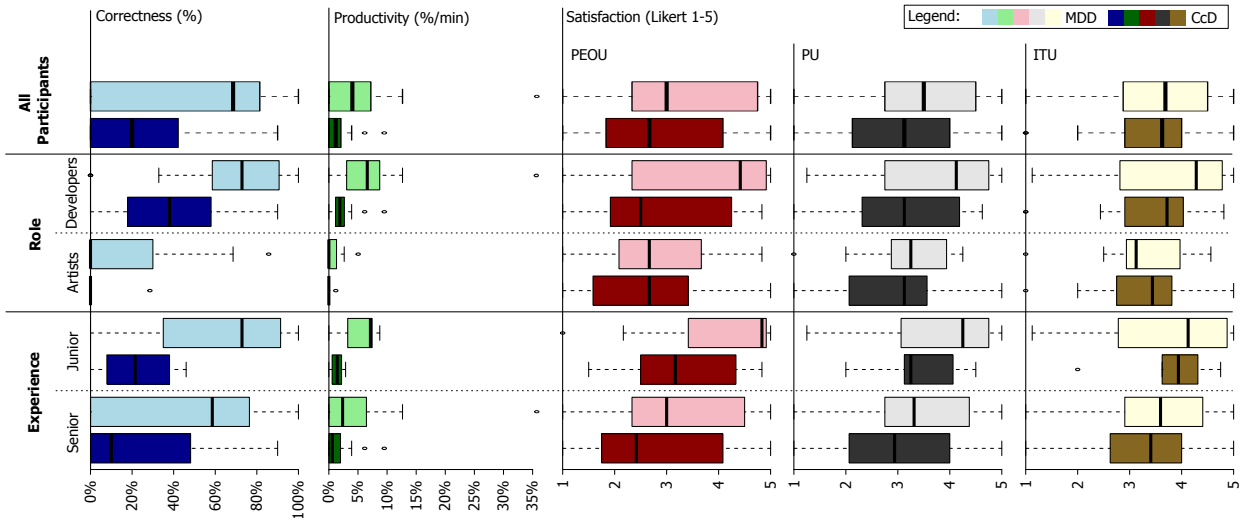


Fig. 6. Boxplots for dependent variables

$SM + Role + Experience + Sequence + Period + Session + SM * Role + SM * Experience + SM * Session\ (1|\ Subj.)$. We choose this model because, in addition to obtaining the highest values for the AIC and BIC fit statistics among all those that verify the normality of the residuals, it reflects the results obtained with simpler statistical models in which the effects of one or two fixed factors, in addition to the main factor, **Specification Method**, were analyzed. It should be noted that, in this type of crossover design, the Period and Sequence factors also represent the combined SM*Sequence and SM*Period effects, respectively. In addition, the use of the Linear Mixed Model (LMM) test assumed that residuals must be normally distributed. 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 a fourth root transformation. For the statistical analysis of this variable with LMM, we used $DV = \sqrt[4]{Efficiency}$. For the rest of the variables, DV is equal to their value.

Table II shows the results of the Type III fixed effects test for each of the response variables or transformations, and for each fixed factor of the statistical model used. P-values indicating significant differences are in italics. The different shades of gray mark the size of the effect of the factor considered (the darker the shade, the larger the effect). According to the results show in Table II, not all the fixed factors included in the statistical models that explain the response variables produce significant changes in them. Moreover, statistically significant changes are only attributable to **Specification Method** in *Correctness*. Conversely, the changes in the dependent variables due to **Role** were statistically significant. However, the effects of the other fixed factors or combinations of these factors on the dependent variables are not statistically significant. It is not possible to statistically rule out the possibility that the effects of these factors on the variables analyzed are due to chance. It is noteworthy that model 0 in equation (1) obtained p-values of $0 < 0.001$ for all the dependent variables considered. However this statistical model does not consider confounding factors that may have affected the changes in the dependent variables.

TABLE II
RESULTS OF THE TYPE III TEST OF FIXED EFFECTS FOR EACH FACTOR OR INTERACTION ACROSS THE DIFFERENT RESPONSE VARIABLES.

	Correctness	*Efficiency*	*PEOU*	*PU*	*ITU*
Specification Method	*0.036*	0.143	0.347	0.457	0.990
Role	*<0.001*	*<0.001*	*0.047*	0.183	0.250
Experience	0.679	0.744	0.363	0.470	0.539
Sequence	0.271	0.480	0.249	0.549	0.689
Period	0.525	0.417	0.276	0.200	0.518
Session	0.585	0.878	0.787	0.710	0.545
SM*Role	0.359	0.821	0.986	0.628	0.379
SM*Session	0.640	0.995	0.830	0.567	0.874
SM*Experience	0.333	0.594	0.885	0.983	0.656

To quantify the differences in the dependent variables due to each fixed factor in the model, we analyzed the Cohen d values. Table III shows the Cohen d values of the dependent variables for all of the fixed factors considered in the statistical analysis. Positive values indicate differences in favor of the first alternative of the factors and negative values indicate differences in favor of the second alternative of the factor. Values indicating a small, medium or large effect due to a factor are highlighted in light, medium and dark gray, respectively. In the case of **Session**, with three alternatives, the table shows the Cohen d values of all two-to-two comparisons of these alternatives. The values are shown in an order triad, where the Cohen d values between alternatives 1 and 2, 2 and 3, and 1 and 3 of the fixed factor are shown in this order.

For *Correctness*, it can be concluded that the differences due to **Specification Method** and **Role** are statistically significant,

TABLE III
COHEN D VALUES FOR EACH FIXED FACTOR ACROSS THE DIFFERENT RESPONSE VARIABLES.

	Correctness	*Efficiency*	*PEOU*	*PU*	*ITU*
Method (MDD vs CCD)	*0.719*	*0.695*	0.356	0.405	0.177
Role (Developer vs Artist)	*1.359*	*0.956*	*0.406*	0.314	0.332
Experience (Junior vs Senior)	0.159	0.034	0.446	0.341	0.293
Group (G1 vs G2)	0.492	0.284	-0.157	-0.054	-0.018
Task (T2 vs T3)	0.226	0.344	0.348	0.397	0.203
Session 1vs2	0.187	-0.283	-0.204	0.0491	-0.031
Session 2vs3	-0.681	0.098	0.375	0.099	0.037
Session 1vs3	-0.465	-0.442	0.169	0.149	0.005

obtained a p-value smaller than 0.05 with the Type III LMM test. All subjects performed the tasks better with MDD than with CCD, and the effect size is medium-large with Cohen *d* values close to 0.8. The effect size of the differences due to **Role** is larger with Cohen *d* values greater than 1. Subjects classified as developers solve the tasks of the experiment more correctly than subjects classified as "Artists", in both MDD and CCD. However, both Developers and Artists solve the tasks better with MDD than with CCD. The comments provided by subjects on the questionnaires and during the focus group offer partial explanations for the observed differences in *Correctness* due to **Role**. Some subjects classified as artists noted that the proposed tasks differed from those they typically performed in their use of Unreal. Others indicated that the inability to utilize AIs or other aids during the experiment hindered their performance in their usual context. Nevertheless, it is important to note that although the artists did not solve their tasks with C++ they were able to tackle them using Blue prints.

For *Efficiency*, the differences between MDD and CCD are medium-large in favor of MDD, however, these differences are not statistically significant in all the statistical models used, and for the selected statistical model obtained a p-value greater than 0.05 with the Type III LMM test. On the other hand, as for *Correctness*, the differences in efficiency due to **role** are statistically significant and large in favor of the subjects classified as developers. Developers solve the tasks of the experiment more efficiency than 'artists', both in MDD and CCD.

For *Satisfaction*, the differences between MDD and CCD in favor of MDD are not statistically significant. Both *PEOU*, *PU* and *ITU* obtain p-values greater than 0.05 with the Type III LMM test. Although there are differences in subject satisfaction in favor of MDD over CCD, these differences are medium for *PEOU*, small for *PU*, and very small for *ITU*. However, the changes in the PEOU variable due to the factor **Role** are medium and statistically significant with Cohen dValue around 0.4 and a p-value of less than 0.05. Developers find it easier to use either method in Unreal than artists do.

The comments of the subjects in the Focus Group can help us interpret this result. The subjects indicated that both techniques are commonly used together in almost all projects, and that each offers its own advantages and disadvantages. The subjects specified that their use depends on who uses them, for what and in what context. They stated that there can be video games 100% made with Blueprints and also video games made 100% with C++, although the ideal is to combine them to take advantage of the best of each one.

The statistical model that best explains the dependent variables includes other fixed factors, such as **Experience**, **Sequence**, **Period**, or **Session**. However, the differences due to these factors are not statistically significant and do not interfere with the differences detected between MDD and CCD. In particular, all subjects, regardless of their professional experience, performed the tasks more correctly with MDD than with CCD. Moreover, there were no significant differences in how correctly or efficiently tasks are performed by subjects with more or less experience.

With these results, we can reject our first null hypothesis. The answer to **RQ1** is affirmative: The **Specification Method** used in Unreal Engine has significant medium-large effects on *Correctness*. However, we cannot reject our second and third null hypotheses. The responses **RQ2** and **RQ3** are negative. We cannot confirm that the specification method used in Unreal Engine has statistically significant effects on *Efficiency* or *Satisfaction*, even though the size of the method effects on *Efficiency* is medium-large.

VI. DISCUSSION

In the context of the widespread Unreal Engine, developers stated in the focus group that they can freely use both models and code to build video games. It is important to highlight that they use models as the main development artifact, not as a mere documentation. In fact, participants argue that it may be possible to build the entire game either using only models or only code.

Both programmers and artists agree that blueprints are more difficult to test than code. They also argue that blueprints are more difficult to merge than code. In addition, they all state that blueprints are worse from a performance standpoint compared to code.

When they do not compare blueprints with code, and talk about blueprints themselves, the participants agree in describing them as difficult to modularize, often they use the word spaghetti to describe them, and even describe them with the adjective chaotic.

On the other hand, the results show that correctness is significantly better with blueprints, and that there is no difference in satisfaction between blueprints and code. In fact, participants agree, that in their opinion, key aspects of video games, such as shaders and AI, are easier to develop with blueprints than with code. Even when asked what % of the video game they would program with C++ or blueprints, there is a consensus on a range of 60% for C++ and 40% for blueprints. No one states 100% for C++ and 0% for blueprints.

The explanation for the success of blueprints in video games could come from their syntax. However, the syntax of blueprints (see Fig.1) relies heavily on humble boxes and

lines. It does not seem that this syntax is revolutionary with respect to classical software modeling languages. No positive (or negative) comments on blueprint syntax surfaced in the focus group either.

One might argue that Blueprints are mainly useful because game development teams include artists with limited technical background. However, our results and focus group feedback show that programmers also benefit from using Blueprints. Statistical analysis revealed that both developers and artists performed better with Blueprints, and the $Role * Method$ interaction was not significant. This indicates that the effectiveness of Blueprints is not limited to non-programmers. Including artists enhances the realism of the study, reflecting the interdisciplinary nature of game development teams.

Focus group insights further clarify the specific benefits of Blueprints. Participants highlighted that their constrained, visual structure helps prevent common programming errors. For example, features like controlled variable manipulation and visual execution flow reduce the likelihood of mistakes that are more easily introduced when writing code. These built-in safeguards may account for the consistent improvements in correctness observed with Blueprints.

Beyond video games, classical software engineering could also unlock the benefits of MDD (see B1–B7 in Section II). However, despite decades of empirical research highlighting these potential benefits, models have not replaced code as the primary development artifact. These decades of research points to several limiting factors, including inadequate UML training (C1), poor tool support (C2), lack of expertise (C3), low perceived usefulness (C4), high investment costs (C5), and the significant training effort required (C6). We think that videogame development (as it is the case with Unreal and Blueprints) can help more classical software engineering to unlock the benefits of MDD.

VII. Threats to Validity

To present the threats to validity, we use the classification proposed by Runeson et al. in [48]. This classification describes the following aspects:

Construct Validity: This aspect of validity defines the extent to which the operational measures under study truly represent the researcher's objective and whether what is being investigated aligns with the research questions. The author bias threat arises if the researchers defining the artifacts can subjectively influence the results they seek. To mitigate this threat regarding *satisfaction*, we used the well-known TAM questionnaire. Regarding performance measurements, widely used metrics were applied: *correctness* and *efficiency*. The single-operation bias threat occurs when there is only one treatment. This study was affected by this threat since the experiment was conducted with a single factor. Under these conditions, generalizing the results must be done with caution.

Internal Validity: This aspect of validity is important when causal relationships are involved. There is a risk that the investigated factor is also affected by an element not considered in the study. The selection threat arises when the experimental results depend on the characteristics of the subjects. This study was affected by this threat because all subjects were professionals from the video game industry. Their professional experience or previous projects may have influenced on the results. Concerning quality measurements, this threat was mitigated by using tasks previously tested in previous works [43].

External Validity: This aspect of validity refers to the extent to which the results can be generalized and their relevance to individuals outside the experiment's setting. The statistical power threat arises when the nature of the data does not allow for the generalization of results. The number of subjects suggests that generalization should be done cautiously. On the other hand, the characteristics of the subjects are noteworthy: the use of professionals makes the contribution significant. The domain threat influence arises when the results are contextualized in a specific domain. Due to its nature, this study is affected by this threat. Generalization could be improved by exploring other domains, such as different game engines and additional examples.

Reliability: This aspect refers to the influence of specific researchers on the data and analysis. Theoretically, if another researcher conducted the experiment, the findings should be the same. The data collection threat arises when data is not gathered consistently across different experimental sessions. This threat was mitigated as all sessions followed the same pattern. In this sense, the experiment is easily replicable with our replication package. The incomplete data threat occurs when some user data is missing. This threat was mitigated as each data capture was performed by one researcher and later the data were reviewed and processed by another researcher.

VIII. Conclusion

Our study shows that Blueprints are not simply tools to assist artists but effective development artifacts in their own right. Both developers and artists benefit from their use, with significant improvements in correctness and no loss in satisfaction compared to code. Although Blueprints pose challenges in testing, modularization, and performance, their constrained, visual structure appears to reduce errors, making them valuable in interdisciplinary teams.

Acknowledgments

We would like to express our gratitude to the subjects who participate in our experiment, without them this work would not have been possible.

This work was supported in part by the Spanish Ministry of Science, Innovation and Universities under the Projects VARNETICA (CNS2023-145422) and PHYLOVAR (PID2024-162114OB-C21, PID2024-162114OA-C22), and in part by the Gobierno de Aragón (Spain) (Research Group T61_23R).

Replication Package

https://doi.org/10.5281/zenodo.15133503

REFERENCES

[1] SlashData, "Global developer population report 2019," https://sdata.me/GlobalDevPop19, 2019, [Online; accessed 21-November-2021].

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

[3] J. Chueca, J. Verón, J. Font, F. Pérez, and C. Cetina, "The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games," *Information and Software Technology*, p. 107330, 2023.

[4] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 1–11.

[5] E. Games, "Unreal engine: The most powerful real-time 3d creation tool," https://www.unrealengine.com, 1998, [Online; accessed 21-November-2021].

[6] E. G. Inc., "Blueprint compiler overview," https://dev.epicgames.com/documentation/en-us/unreal-engine/compiler-overview-for-blueprints-visual-scripting-in-unreal-engine?application_version=5.3, 2024, [Online; accessed 25-March-2024].

[7] K. Krogmann and S. Becker, "A Case Study on Model-Driven and Conventional Software Development : The Palladio Editor," *Software Engineering*, 2007.

[8] P. E. Papotti, A. F. Do Prado, W. L. de Souza, C. E. Cirilo, and L. F. Pires, "A quantitative analysis of model -driven code generation through software experimentation," in *International Conference on Advanced Information Systems Engineering*. Springer, 2013, pp. 321–337.

[9] N. C. Marko, G. Liebel, D. Sauter, A. Lodwich, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering for embedded systems in practice," *Research reports in software engineering and management*, pp. 1–48, 2014.

[10] D. Akdur, V. Garousi, and O. Demirörs, "A survey on modeling and model-driven engineering practices in the embedded software industry," *Journal of Systems Architecture*, vol. 91, pp. 62–82, 2018.

[11] T. Kapteijns, S. Jansen, S. Brinkkemper, H. Houet, and R. Barendse, "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.

[12] W. Heijstek and M. R. V. Chaudron, "Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process," in *Conference Proceedings of the EUROMICRO*, 2009.

[13] Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Evaluating the benefits of model-driven development," in *International Conference on Advanced Information Systems Engineering*. Springer, 2020, pp. 353–367.

[14] Y. Martínez, C. Cachero, and S. Meliá, "MDD vs. traditional software development: A practitioner's subjective perspective," in *Information and Software Technology*, 2013.

[15] J. I. P. Navarrete, O. Dieste, B. Marín, S. España, S. Vegas, O. Pastor, and N. Juristo, "Evaluating model-driven development claims with respect to quality: A family of experiments," *IEEE Transactions on Software Engineering*, 2018.

[16] M. R. Chaudron, W. Heijstek, and A. Nugroho, "How effective is UML modeling?" *Software & Systems Modeling*, vol. 11, no. 4, pp. 571–580, 2012.

[17] T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *Journal of Systems and Software*, vol. 95, pp. 176–193, 2014.

[18] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 295–310, 2004.

[19] M. Grossman, J. E. Aronson, and R. V. McCarthy, "Does UML make the grade? Insights from the software development community," *Information and Software Technology*, vol. 47, no. 6, pp. 383–397, 2005.

[20] B. Dobing and J. Parsons, "How UML is used," *Communications of the ACM*, vol. 49, no. 5, pp. 109–113, 2006.

[21] L. T. W. Agner, I. W. Soares, P. C. Stadzisz, and J. M. SimãO, "A Brazilian survey on UML and model-driven practices for embedded software development," *Journal of Systems and Software*, vol. 86, no. 4, pp. 997–1005, 2013.

[22] H. Störrle, "How are Conceptual Models used in Industrial Software Development? A Descriptive Survey," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 160–169.

[23] B. Anda, K. Hansen, I. Gullesen, and H. K. Thorsen, "Experiences from introducing UML-based development in a large safety-critical project," *Empirical Software Engineering*, vol. 11, no. 4, pp. 555–581, 2006.

[24] M. Staron, "Adopting model driven software development in industry–a case study at two companies," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 57–72.

[25] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[26] R. Jolak, T. Ho-Quang, M. R. Chaudron, and R. R. Schiffelers, "Model-based software engineering: A multiple-case study on challenges and development efforts," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018, pp. 213–223.

[27] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Relevance, benefits, and problems of software modelling and model driven techniques—a survey in the italian industry," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2110–2126, 2013.

[28] J. I. Trasobares, Á. Domingo, L. Arcega, and C. Cetina, "Evaluating the benefits of software product lines in game software engineering," in *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A*, 2022, pp. 120–130.

[29] J. Chueca, J. I. Trasobares, Á. Domingo, L. Arcega, C. Cetina, and J. Font, "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*, vol. 205, p. 111824, 2023.

[30] I. Roca, Á. Domingo, Ó. Pastor, C. Cetina, and L. Arcega, "Comparing mdd and ccd in the bug localization context: An empirical evaluation in video games," in *International Conference on Advanced Information Systems Engineering*. Springer, 2024, pp. 581–595.

[31] S. Martínez-González, F.-J. Azcondo-San-Segundo, J.-A. Díaz-Severiano, and V. Gomez-Jauregui, "A comparative analysis of code-based versus visual programming approaches for extended reality (xr) and bim," in *Advances on Mechanics, Design Engineering and Manufacturing V*, C. Manchado del Val, R. Miralbes Buil, G. Peris Fajarnés, M. Moncho Santonja, C. Rizzi, and L. Roucoules, Eds. Cham: Springer Nature Switzerland, 2025, pp. 543–555.

[32] E. F. do Prado and D. Lucredio, "A flexible model-driven game development approach," in *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software*, 2015, pp. 130–139.

[33] M. Derakhshandi, S. Kolahdouz-Rahimi, J. Troya, and K. Lano, "A model-driven framework for developing android-based classic multiplayer 2d board games," *Automated Software Engineering*, vol. 28, no. 2, p. 7, 2021.

[34] E. Fanouraki, "Did you know that 60% of game developers use game engines?" https://www.slashdata.co/blog/did-you-know-that-60-of-game-developers-use-game-engines, 2022, [Online; accessed 05-December-2023].

[35] U. Technologies, "Unity real-time development platform — 3d, 2d vr & ar engine," https://unity.com, 2005, [Online; accessed 21-November-2021].

[36] Crytek, "Cryengine — the complete solution for next generation game development by crytek," https://www.cryengine.com, 2002, [Online; accessed 21-November-2021].

[37] E. G. Inc., "Real-time round-up: the state of interactive 3d," https://www.unrealengine.com/en-US/blog/real-time-round-up-the-state-of-interactive-3d, 2023, [Online; accessed 11-December-2023].

[38] ——, "About epic games," https://www.epicgames.com/site/en-US/about, 2024, [Online; accessed 13-March-2024].

[39] activeplayer.io Game Statistics Authority, "Fortnite live player count and statistics," https://activeplayer.io/fortnite/, 2023, [Online; accessed 11-December-2023].

[40] V. R. Basili and H. Dieter Rombach, "The tame project: Towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, 1988.

[41] D. L. Moody, "The method evaluation model: a theoretical model for validating information systems design methods," *ECIS 2003 proceedings*, p. 79, 2003.

[42] S. Vegas, C. Apa, and N. Juristo, "Crossover designs in software engineering experiments: Benefits and perils," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 120–135, 2015.

[43] J. Verón, C. Pérez, C. Calero, M. Moraga, F. Pérez, and C. Cetina, "A comparative analysis of energy consumption between visual scripting models and c++ in unreal engine: Raising awareness on the importance of green mdd," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 114–125.

[44] B. T. West, K. B. Welch, and A. T. Galecki, *Linear mixed models: a practical guide using statistical software*. Chapman and Hall/CRC, 2014.

[45] E. I. Karac, B. Turhan, and N. Juristo, "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.

[46] Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Comparing uml-based and dsl-based modeling from subjective and objective perspectives," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 483–498.

[47] J. Cohen, "Statistical power for the social sciences," *Hillsdale, NJ: Laurence Erlbaum and Associates*, 1988.

[48] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8