Comparing Software Product Lines and Clone and Own for Game Software Engineering under two paradigms: Model-Driven Development and Code-Driven Development

Jorge Chueca*, Jose Ignacio Trasobares, África Domingo, Lorena Arcega, Carlos Cetina, Jaime Font

Universidad San Jorge. SVIT Research Group
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain

Abstract

Game developers often face more challenges when reusing code compared to non-game developers, and Software Product Lines (SPLs) have been successful in addressing this issue. In this study, we compare different approaches to code reuse in Classic Software Engineering (CSE) and Game Software Engineering (GSE) through a commercial video game called Kromaia. We specifically focus on two development paradigms: Model-Driven Development (MDD) and Code-Driven Development (CDD). We conduct an empirical evaluation where subjects develop game elements using two approaches: Clone and Own (CaO) and SPLs.The results show that game elements developed using SPLs are more correct (over 23%) than those developed with CaO in both MDD and CDD paradigms. In CDD, there are significant improvements in efficiency (51%) and satisfaction (13%) when using SPLs compared to CaO. However, no improvements are observed when working under MDD. The impact of using SPLs or CaO is greater in CDD than in MDD for game developers. Our findings suggest that SPLs in GSE may have a different role compared to their traditional role in CSE. Specifically, SPLs can be valuable in balancing game difficulty or generating new video game content, such as the one present in the bosses of the game.

Keywords: Empirical evaluation, Software Product Line Engineering, Game Software Engineering, Model-Driven Development, Code-Driven Development

1. Introduction

Nowadays, the video game industry is one of the fastest-growing industries in the world. According to a 2019 report [46], the total number of active software developers is 18.9M. The same report indicates that 8.8M of these active developers have worked or are working on video games. This means that almost half of the active developers have been involved in video game development at some point. Video game development has differences from classic software development [26, 36]. Hence, it is important to note that Game Software Engineering (GSE) aligns with Classic Software Engineering (CSE), sharing many similarities rather than significant differences [1]. One difference is that game developers often face more challenges when reusing code compared to non-game developers and do not have standards to avoid technical debt [8, 42].

The majority of video games are developed using game engines, which are development environments that integrate two engines for graphics and physics and a set of tools to accelerate development. The most popular are Unity [48] and Unreal Engine [20], but it is also possible for a studio to make its own engine (e.g., CryEngine [12]). Game engines allow video game developers to create content directly using code (e.g., C++) or software models.

Software models raise the abstraction level using terms that are much closer to the problem domain. The Model-Driven Development (MDD) [44] paradigm focuses on the creation of models that can then be translated into source code. This means that developers can focus on the game content itself, abstracting from the implementation details. In addition, game engines also allow working under the traditional Code-Driven Development (CDD) paradigm. Working at the code level allows developers to have more control over the implementation details when needed.

The use of Software Product Lines (SPLs) has proven to be effective in CSE for developing different types of software at a lower cost, in less time, and with higher quality [41]. That is why there are recent research efforts that propose applying SPLs in the domain of video games [29, 32, 45]. However, recent research [40] has provided evidence that game developers perceive more difficulties than non-game developers when reusing code. Contrary to

*Corresponding author
Email addresses: jchueca@usj.es (Jorge Chueca), jtrasobaresibor@acm.org (Jose Ignacio Trasobares), adomingo@usj.es (África Domingo), larcega@usj.es (Lorena Arcega), ccetina@usj.es (Carlos Cetina), jfont@usj.es (Jaime Font)

a systematic development method exists Clone and Own (CaO), it involves the informal ad-hoc practice of extracting reusable artifacts (e.g., code) from the software and duplicating them in other parts of the program [19]. This practice is commonly known as *copy and paste.*

This paper is an extension of our previous work [50] in which we performed an empirical evaluation to compare two development approaches, CaO and an SPL under the MDD paradigm. The evaluation was performed using Kromaia, a commercial video game that other authors have previously analyzed [6, 17, 16, 5]. The 28 participants had to develop two final bosses of Kromaia, one using CaO and another one using a compositional SPL. The results were compared in terms of correctness, efficiency, and user satisfaction. Although the bosses developed with the SPL were more correct than those developed with CaO, the results did not indicate significant changes in efficiency or satisfaction.

The evaluation of the previous work focused only on the MDD paradigm; in this work, we extend the empirical evaluation of our previous work to include a new development paradigm in the analysis, the CDD paradigm. Although the MDD paradigm is being used by video game developers, working at the level of source code is more common in GSE. With this work, we want to determine if the benefits of using an SPL rather than CaO when developing video game content under the MDD paradigm discovered in the previous work still apply under the CDD paradigm. Furthermore, we will analyze the aggregated results from the MDD and CDD paradigms to determine the effect of the paradigm on the results.

To this end, 51 new subjects were recruited to develop two final bosses of Kromaia in C++. Each boss was created following a different development approach: one using CaO, reusing code from existing bosses and libraries, and another using an annotative SPL that derives final bosses in C++ code. Then, the results were analyzed and compared following the same indicators of correctness, efficiency, and user satisfaction employed in our previous work.

The empirical evaluations we perform are presented from the perspective of *controlled experiments* [54] in which humans apply different treatments to one or more objects of study in a laboratory setting. The use of a laboratory setting allows greater control over the manipulation and measurements of the variables involved in the study in order to test hypotheses or determine causal relationships. We acknowledge that some readers may refer to this study as a *case study.* However, we use the Wohlin et al. terminology for empirical studies [54]. In this context, the term *case study* does not fit our empirical evaluation because although we use a real video game to extract the objects to evaluate the approaches SPL & CaO, our study does not take place in a real-world context but in laboratory settings and the focus of the study is not the video game analysis itself.

Our results suggest that the SPLs obtain better results than CaO, especially under the CDD paradigm. However, the SPL approach in GSE does not have the same effects that have made SPLs attractive to CSE. In light of our results, we may need to rethink the role of SPLs for GSE, treating GSE as a different domain than CSE when dealing with SPLs. Our work suggests new research directions for SPL in GSE. Specifically, our work reveals that SPLs can be relevant to the particularities of GSE: generating new video game content (one of the hot topics of video game research), making software accessible to multidisciplinary teams, or balancing difficulty (one of the seminal problems of video games).

The remainder of the paper is structured as follows. Section 2 presents the video game Kromaia and the two development paradigms studied (MDD and CDD). Section 3 describes our experiment. Section 4 presents the results obtained. Section 5 discusses the findings. Section 6 describes the threats to validity. Section 7 examines the related work of the area. Finally, Section 8 concludes the paper.

2. Kromaia and its paradigms

This section presents Kromaia[1] and the multiple processes available to generate bosses applying the two development approaches (CaO or SPL) under the two paradigms studied (MDD and CDD).

Kromaia is a three-dimensional space game created by Kraken Empire, in which the player's spaceship flies from a starting point to an ending point, reaching the goal of each level before being destroyed. Throughout each level, the player has to explore floating structures, avoid asteroids and find items. Along the way, the player will encounter basic enemies trying to damage the player's spaceship by firing projectiles. If the player reaches the destination, the final boss corresponding to that level appears and must be defeated in order to complete the level.

Figure 1 presents an overview of the architecture of Kromaia. The top layer (content creation) allows the developers to add new content to the video game in two different ways, using models (SDML) or source code (C++):

- SDML: The Shooter Definition Modeling Language, created by the developers of the game, allows the creation of any type of element that is present in the game. It will be used in the experiment under the MDD paradigm.
- C++: The C++ programming language can also be used to create content for the game. Game elements created in C++ must follow some structure rules and patterns (e.g., inherit from a specific class) in order to be properly integrated with the rest of the game.

[1]Kromaia Omega - Launch — PlayStation 4: `https://youtu.be/EhsejJBp8Go`

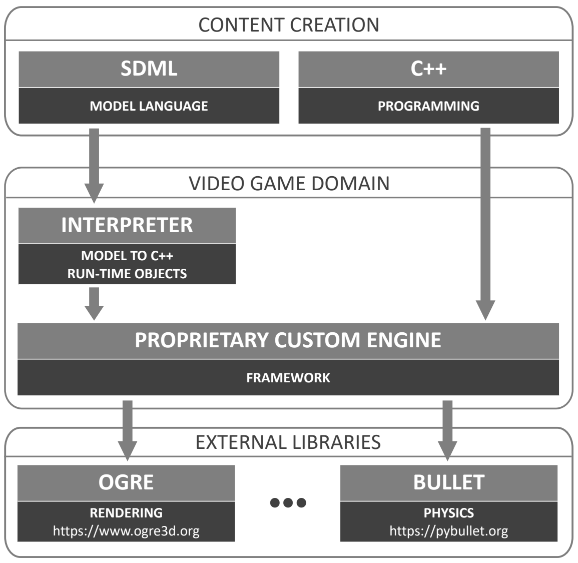


Figure 1: Architecture of Kromaia. Adapted from [6]

The middle layer (video game domain) contains the core of the game and its main functionality. When the content is provided as an SDML model, it will be interpreted at run-time and transformed into a C++ run-time object. When the content is provided in the form of source code, it will interact directly with the proprietary custom engine. The engine contains the base classes and functionality used by the game elements. The developers of the game also created an Application Programming Interface (API) to facilitate the creation of new content and its integration with the rest of the game.

The bottom layer (external libraries) contains the set of libraries used for specific purposes, such as the rendering of the game (Ogre) or the management of the physics (Bullet). The engine is built upon those libraries and will interact with them when needed.

Below, we present how the two development paradigms studied (MDD and CDD) are realized in the Kromaia architecture.

2.1. The MDD paradigm in Kromaia

Figure 2 shows the main artifacts used under the MDD paradigm for developing final bosses in Kromaia. Bosses are specified using the Shooter Definition Modeling Language (SDML) [6]. SDML is a Domain-Specific Language (DSL) that defines components that appear in video game entities: the anatomical structure (including which parts are used in it, their physical properties, and how they are connected); the amount and distribution of vulnerable parts, weapons, and defenses in the structure/body of the character; and the movement behaviours associated to the whole body or its parts. This modeling language has concepts such as hulls, links, weak points, and weapons.

The top part of Figure 2 shows a simplified version of the SDML metamodel and its concrete syntax. The complete metamodel contains more than 20 concepts, over 20 relationships, and more than 60 properties [6]. However, this simplified version is complete enough to understand the elements that compose the structure of a boss and their relationships with each other. The figure depicts the concrete syntax of each of the elements. Overall, a boss is composed of hulls and links. Each link joins two hulls. A hull can contain weak points that the player must attack, weapons such as cannons, lasers, or spikes, and shields that protect weak points and weapons.

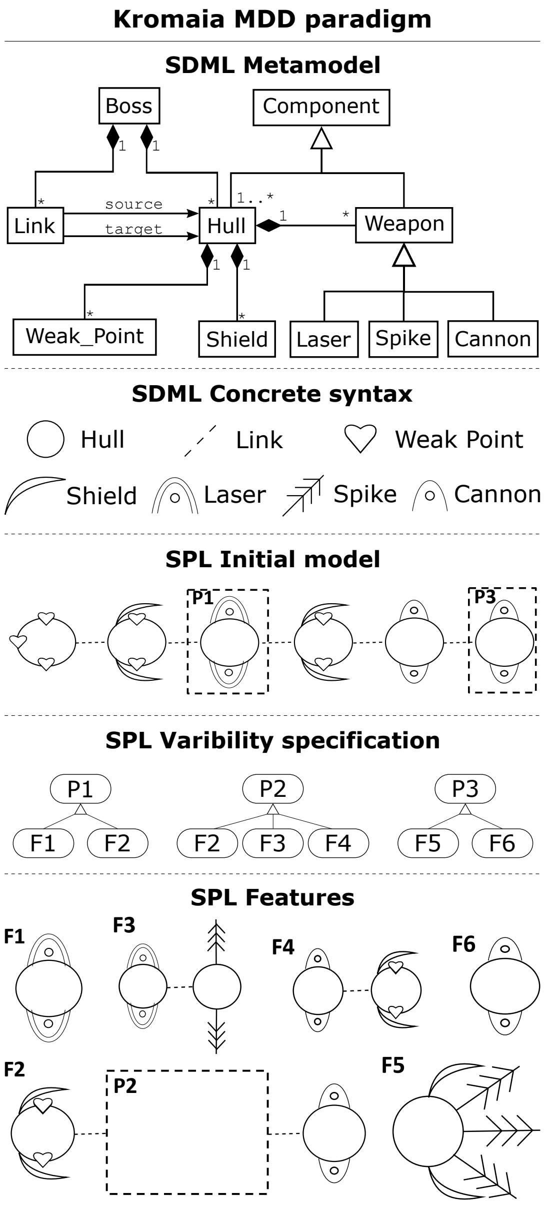
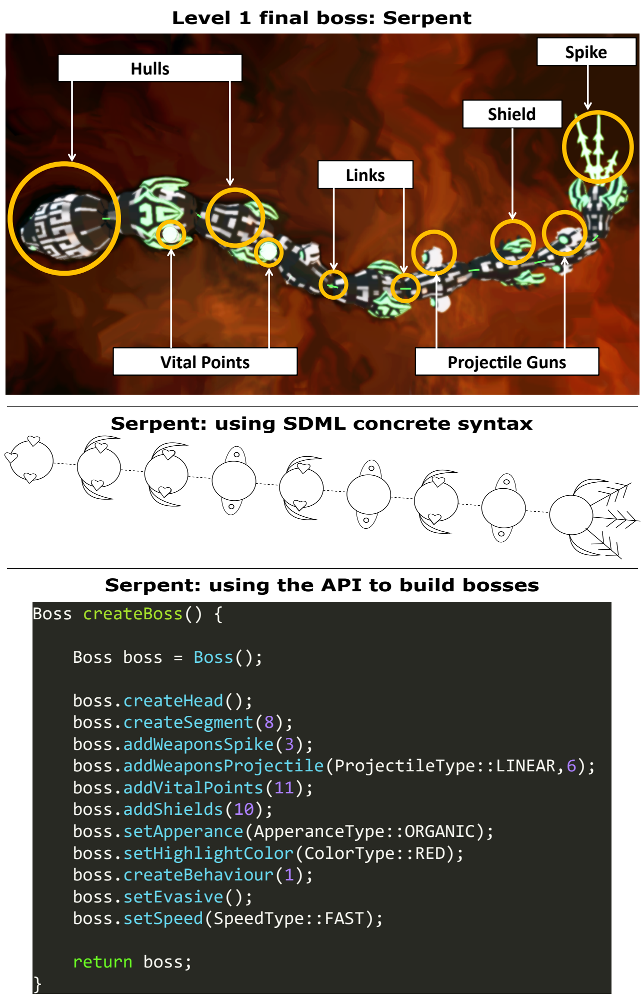


Figure 2: Kromaia MDD paradigm: the Shooter Definition Modeling Language (SDML) (Simplified version) and a subset of the SPL used to develop models of bosses under the MDD paradigm.

An actual example of a final boss of Kromaia is presented in the top part of Figure 3. The Serpent is the final boss that the player must defeat in order to complete Level 1. The middle part of the figure shows the model of the boss using the SDML concrete syntax. This boss is


Level 1 final boss: Serpent
Hulls
Spike
Shield
Links
Vital Points
Projectile Guns
Serpent: using SDML concrete syntax
Serpent: using the API to build bosses

```
Boss createBoss() {

    Boss boss = Boss();

    boss.createHead();
    boss.createSegment(8);
    boss.addWeaponsSpike(3);
    boss.addWeaponsProjectile(ProjectileType::LINEAR,6);
    boss.addVitalPoints(11);
    boss.addShields(10);
    boss.setApperance(ApperanceType::ORGANIC);
    boss.setHighlightColor(ColorType::RED);
    boss.createBehaviour(1);
    boss.setEvasive();
    boss.setSpeed(SpeedType::FAST);

    return boss;
}
```

Figure 3: Example of a final boss in Kromaia.

composed of a series of nine hulls that are linked in a line. The first hull is the head, which includes three weak points and will drive the movement of the boss. The next seven hulls contain either two vital points covered with shields or two cannons. The last hull is the tail, containing two shields and three spikes that are used to hit the player.

Under the MDD paradigm, when bosses are created using the CaO development approach, the developer has access to a set of SDML models that have been created in the past (such as the serpent in the middle part of Figure 3). The developer can use them to serve as the basis for the new boss being created or can reuse parts of the existing models to build the new boss. For example, the developer could copy the existing serpent model and add some new weapons to one of the existing hulls or copy the last two hulls and connect them to create a two-tailed serpent boss.

Under the MDD paradigm, when bosses are created using the SPL development approach, the developer has access to a feature-oriented SPL that can be used to generate SDML models. The SPL follows a compositional approach [25]. The developer will begin with an initial model and perform some substitutions of model fragments using a library of model fragments and following the rules specified by the variability specification.

The middle part of Figure 2 shows a subset of the assets of the SPL for developing Kromaia bosses. The figure shows one of the initial models that can be used to create new bosses. The bottom of Figure 2 shows some feature models that represent the variability specification and some of the features present in the library of the SPL. Each feature corresponds to a model fragment that is expressed using the concrete syntax of SDML.

The domain experts of Kromaia defined the variability specification. The elements denoted with a P represent variation points that can be substituted by a feature, F, following the variability specification. For example, the variability specification that is shown in the first position ($P1$) indicates that it can be substituted with the model fragments $F1$ or $F2$. Furthermore, some features can include variation points that must be fulfilled with another feature. For example, feature $F2$ has a variation point, $P2$, which has to be substituted following its variability specification (by $F2$, $F3$ or $F4$).

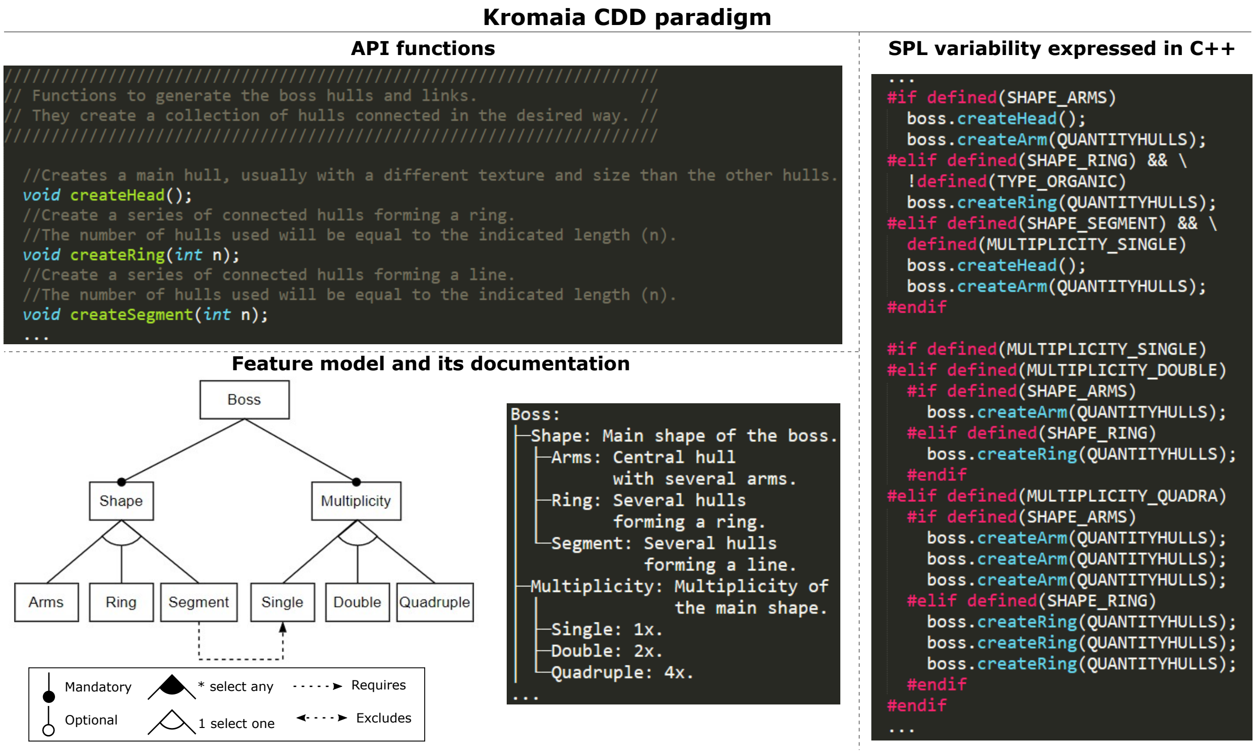
2.2. The CDD paradigm

Figure 4 shows the main artifacts used when creating a boss under the CDD paradigm. The bottom part of Figure 3 shows an example of a boss created under the CDD paradigm, where the final bosses are specified using the C++ programming language and the API of Kromaia [6].

The API contains a set of base classes that can be extended when creating new content. It also contains functionality shared by many elements of the video game. The top part of Figure 4 shows a subset of the API for creating bosses in Kromaia. It is structured in different sections that include: functions to create a series of hulls linked together following different forms (rings, segments, squares) and different sizes; functions to add weapons to existing hulls and to manage the properties of those weapons (e.g., type of projectile used); functions to include vital points and shields to the different hulls; and functions to determine the appearance (e.g., predominant color for the hulls) and behaviour (e.g., different IA parameters that determine the actions taken in the game) of the boss being created.

Under the CDD paradigm, when bosses are created using the CaO development approach, the developer has access to the API, its documentation, and a set of source codes of other bosses that have been created in the past (such as the Serpent shown at the bottom part of Figure 3). The developer will use them in the same way as in the MDD paradigm by serving as the basis for the new boss being created, reusing parts of the existing code to build the new boss, adding new lines of code, or calling the functions in the API.

Under the CDD paradigm, when bosses are created using the SPL development approach, the developer has access to an annotative SPL [25] that can be used to generate



```
/////////////////////////////////////////////////////////////////////
// Functions to generate the boss hulls and links.                 //
// They create a collection of hulls connected in the desired way. //
/////////////////////////////////////////////////////////////////////

  //Creates a main hull, usually with a different texture and size than the other hulls.
  void createHead();
  //Create a series of connected hulls forming a ring.
  //The number of hulls used will be equal to the indicated length (n).
  void createRing(int n);
  //Create a series of connected hulls forming a line.
  //The number of hulls used will be equal to the indicated length (n).
  void createSegment(int n);
  ...
```

```
Boss:
├─Shape: Main shape of the boss.
│ ├─Arms: Central hull
│ │       with several arms.
│ ├─Ring: Several hulls
│ │       forming a ring.
│ └─Segment: Several hulls
│            forming a line.
└─Multiplicity: Multiplicity of
                the main shape.
  ├─Single: 1x.
  ├─Double: 2x.
  └─Quadruple: 4x.
...
```

```
...
#if defined(SHAPE_ARMS)
  boss.createHead();
  boss.createArm(QUANTITYHULLS);
#elif defined(SHAPE_RING) && \
  !defined(TYPE_ORGANIC)
  boss.createRing(QUANTITYHULLS);
#elif defined(SHAPE_SEGMENT) && \
  defined(MULTIPLICITY_SINGLE)
  boss.createHead();
  boss.createArm(QUANTITYHULLS);
#endif

#if defined(MULTIPLICITY_SINGLE)
#elif defined(MULTIPLICITY_DOUBLE)
  #if defined(SHAPE_ARMS)
    boss.createArm(QUANTITYHULLS);
  #elif defined(SHAPE_RING)
    boss.createRing(QUANTITYHULLS);
  #endif
#elif defined(MULTIPLICITY_QUADRA)
  #if defined(SHAPE_ARMS)
    boss.createArm(QUANTITYHULLS);
    boss.createArm(QUANTITYHULLS);
    boss.createArm(QUANTITYHULLS);
  #elif defined(SHAPE_RING)
    boss.createRing(QUANTITYHULLS);
    boss.createRing(QUANTITYHULLS);
    boss.createRing(QUANTITYHULLS);
  #endif
#endif
...
```

Figure 4: Kromaia CDD paradigm: a subset of the API functions and a subset of the SPL used to develop bosses under the CDD paradigm.

the source code of the bosses (see the bottom and right part of Figure 4). The developer will derive new products through the feature model, creating a configuration that will then be used to generate the source code. The developer has access to the feature model itself, the documentation of each of the features present in it, and the annotated source code that is being configured through the feature model.

Figure 4 shows part of the feature model driving the SPL under the CDD paradigm. It determines the different features that can be selected when creating a boss from the SPL and some constraints about the choices that can be made (e.g., having a segment shape implies a single multiplicity). The documentation includes a description of each of the features that can or cannot be included in the boss being created. When a configuration of a product is done, it will be used to generate the source code of the boss following an annotative approach. Each of the features will produce a set of *define* macros that will be used to pre-compile a C++ source code that includes the information about all the bosses (see Figure 4, right). Depending on the features selected and the *#define* macros generated, a different set of C++ lines will be included in the resulting C++ code. For example, if the feature *Arms* and the feature *Quadruple* are selected, the macro *define SHAPE_ARMS* and *define MULTIPLICITY_QUADRA* will be used to pre-compile the source code, resulting in the line *boss.createArm(x)* being included four times in the final source code (the *QUANTITYHULLS* is defined by another feature that is related to the size of the boss being created).

3. Experiment design

This section presents the experimental design of both experiments. It includes information about the objectives, the variables studied, the research questions, the design chosen, the participants recruited, the experimental objects used during the experiments, the procedure followed in each execution of the experiments, and the statistical analysis selected for the treatment of the data collected.

3.1. Objectives

According to Wohlin's guidelines [54] for reporting software engineering experiments, we have organized our research objectives using the Goal Question Metric template for goal definition [4].

Our goal is to **analyze** different development approaches **for the purpose of** comparison, **with respect to** the correctness, efficiency, and user satisfaction of the software constructed, **from the point of view of** the experience of the developer (inexperienced or experienced), **in the context of** creating software for video games under two different development paradigms: MDD and CDD.

3.2. Variables

In this study, the factor under investigation is the Development Approach (DA). We investigate two alternatives of the factor: create the final bosses of a video game using CaO or using an SPL.

To evaluate the effects of the use of the different approaches, we selected Correctness and Efficiency (related to the subjects' performance) as objective dependent variables, and user Satifaction (related to the subjects' perception) as subjective dependent variable.

We measured Correctness using a correction template, which was applied to the bosses developed by the subjects after the experiment. Their values range from 0 to 100 and represent the percentage of points obtained according to the correction template.

To calculate Efficiency, we measured the time employed by each subject to finish the task, using the start and end times of each task. Efficiency is the ratio of Correctness to time spent (in minutes) to perform a task.

We measured Satisfaction using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [37]. We decompose Satisfaction into three subjective dependent variables as follows:

- Perceived Ease of Use (PEOU), the degree to which a person believes that learning and using a particular DA would require less effort.
- Perceived Usefulness (PU), the degree to which a person believes that using a particular DA will increase performance.
- Intention To Use (ITU), the degree to which a person intends to use a DA.

Each of these variables corresponds to specific items in the TAM questionnaire. We average the scores obtained for these items to obtain the value for each variable.

To take into account the effects that the different development paradigms could have on the DA, we considered the Paradigm blocking factor with two alternatives: CDD and MDD. We carried out two experiments with the same design, each one based on a different development paradigm. The first experiment involves GSE tasks under the MDD paradigm, while the second experiment involves GSE tasks under the CDD paradigm.

3.3. Research questions and hypotheses

The research questions and null hypotheses are formulated as follows:

RQ1 - Does the DA used to create software for video games impact the Correctness of the software? The corresponding null hypothesis is $H_{0,C}$: The DA used for creating software for video games does not have an effect on Correctness.

RQ2 - Does the DA used to create software for video games impact the Efficiency of developers? The null hypothesis for Efficiency is $H_{0,E}$: The DA does not have an effect on Efficiency.

RQ3 - Is user satisfaction different when developers use different DA to create software for video games? To answer this question, we formulated three hypotheses based on the variables *PEOU, PU*, and ITU with their corresponding null hypotheses:

$H_{0,PEOU}$ - The DA does not have an effect on PEOU.

$H_{0,PU}$ - The DA does not have an effect on PU.

$H_{0,ITU}$ - The DA does not have an effect on ITU.

RQ4 - Are the answers to RQ1, RQ2, and RQ3 the same when using different development paradigms (MDD and CDD)? To address this question, we will answer RQ1, RQ2, and RQ3 for each of the two paradigms considered.

3.4. Design

For the two experiments of this study, we use a factorial crossover design with two periods, using different tasks (T1 and T2) for each period. The subjects are randomly divided into two groups (G1 and G2). In the first period of the experiment, all of the subjects perform T1, with subjects in G1 using CaO and subjects in G2 using SPL. Then, in the second period of the experiment, all of the subjects perform T2, with subjects in G1 using the SPL and subjects in G2 using CaO.

The repeated measures design increases the sensitivity of the experiment [52]: the observation of the same subject using the two alternatives controls between-subject differences, improving experiment robustness regarding variation among subjects. By using two different sequences for each group (G1 uses CaO first and SPL second; G2 uses SPL first and CaO second) and different tasks, the design counterbalances some of the effects that can be caused by the order in which development approaches are used (i.e., learning effect, fatigue).

Before conducting each experiment, to verify the experiment design, we conducted a pilot study with two subjects. The pilot study allowed us to estimate the time needed to perform the tasks and complete the questionnaires, detect typographical and semantic errors, and test the instruments used to collect the data and conduct the experiment. The two subjects were different for each pilot study, and they did not participate in the experiments.

3.5. Participants

We selected the subjects using convenience sampling [54] via an open call to volunteers. A total of 79 subjects with different knowledge about programming, modeling, and video game development performed the experiment; 28 of them performed the experiment under the MDD paradigm and 51 performed the experiment under the CDD paradigm. Table 1 presents detailed demographic data about the subjects.

Each execution of the experiment was conducted by two instructors and one expert in video game development. The expert provided information about the Kromaia framework and examples of how to create bosses using CaO and the SPL under the paradigm studied in the experiment. During the experiment, one of the instructors gave instructions and managed the focus groups, while the other clarified doubts for the subjects and took notes.

3.6. Experimental objects

The tasks of our experiments consisted in developing two final bosses of Kromaia from a gameplay video showing its structure and behavior. A video game software engineer involved in the development of Kromaia and a researcher designed the tasks, balanced the complexity (to ensure similar difficulty across tasks), and prepared the correction templates.

The subjects got access to training material with examples of how to create a boss under the paradigm being studied (MDD or CDD) using CaO or the SPL. Depending on the DA and paradigm being used, the subjects got access to different artifacts to create the bosses (see Section 2).

For the data collection, we prepared a set of two forms for each experiment (one for each experimental sequence) with the following sections: *(I)* an informed consent statement that the subjects had to review and accept voluntarily, which clearly explained what the experiment consisted of and what will be the treatment given to personal data; *(II)* a demographic questionnaire to characterize the sample; and *(III)* a specific questionnaire to collect the subjects' responses during the experiment (their tasks, their times, and their answers to the satisfaction questionnaire).

The experimental objects used in these experiments (the training material, the tasks, and the forms used for the questionnaires), as well as the results and the statistical analysis, are available to the reader[2].

3.7. Experimental procedure

Each of the experiments performed for this study was conducted on two different days. On one of the days, the experiment was conducted online with professionals (experienced subjects). On the other day, the experiment was conducted face-to-face with the group of students (inexperienced subjects). All of the sessions were scheduled for a duration of one hour and 45 minutes and were carried out following the experimental procedure described below:

1. ($\sim$ 5 min) An instructor explained the parts of the session and clarified that it was not a test of their abilities.
2. ($\sim$10 min) The subjects attended a tutorial about the video game bosses to be created and how to use CaO or SPL to create these bosses under the paradigm studied in the experiment (MDD or CDD).
3. ($\sim$5 min) The subjects received clear instructions on how to access the form for the experiment and the artifacts needed to complete the task. The subjects were randomly divided into two groups (G1 and G2) and each group got access to the corresponding form.
4. ($\sim$5 min) The subjects accessed the form and then read and confirmed having read the information about the experiment, the data treatment of their personal information, and the voluntary nature of their participation before accessing the questionnaires and tasks of the experiment.
5. ($\sim$5 min) The subjects completed a demographic questionnaire.
6. ($\sim$30 min) The subjects performed the first task. The subjects from G1 had to use the CaO approach to create a given boss of the video game, and the subjects from G2 had to create the same boss but using the SPL. After submitting their solutions, the subjects completed a satisfaction questionnaire about the approach used for development.
7. ($\sim$30 min) The subjects performed the second task, which has the creation of another boss. This time, the subjects from G1 created it using the SPL approach, while the subjects from G2 created it using the CaO approach. Then, the subjects completed the satisfaction questionnaire.
8. ($\sim$15 min) A focus group interview about the tasks was conducted by the instructors.

3.8. Analysis procedure

To analyze the data extracted from the experiments, we performed a statistical study of the aggregated data from the two experiments, and we also performed a statistical study of the data extracted from each experiment separately. The analysis procedure was the same for all three data sets. We chose the Linear Mixed Model (LMM) [53] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows us to study the effects of factors that intervene in a crossover design (period and sequence) and the effects of other blocking factors (Paradigm or Experience) [52].

In the hypothesis testing, we applied the Type III test of fixed effects with unstructured repeated covariance. Type III is the default test, which enables LMM to produce the exact F-values and p-values for each dependent variable and each fixed factor. The assumption for applying LMM is the normality of the residuals of the dependent variables. To verify this normality, we used Kolmogorov-Smirnov tests as well as visual inspections of the histogram and normal Q-Q plots.

In this study, DA was defined as a fixed-repeated factor to identify the differences between using CaO or SPL, and the subjects were defined as a random factor $(1|Subj.)$ to reflect the variability among subjects. The dependent variables (DV) for this test were Correctness and Efficiency, and the three subjective variables related to Satisfaction: PEOU, PU, and ITU.

To analyze the potential effects of the development paradigm to determine the variability due to DA in the dependent variables, in the statistical models used for analyzing the aggregated data, we considered the Paradigm fixed factor and the combination of this factor with DA. In the statistical models used to analyze the three data sets, we also considered Period and Sequence as fixed factors

[2]Available at `http://svit.usj.es/SPLvsCAO-MDD-CDD`

to take into account the potential effects of factors that intervene in a crossover design. In order to take into account the potential effects of the subjects' experience to determine the variability in the dependent variables, we also considered Experience as a fixed factor and its combinations with the other fixed factors considered in the statistical models (DA, Paradigm, Sequence, and Period).

We tested different statistical models to find out which factors, in addition to DA, could best explain the changes in the dependent variables. Some of these statistical models are described mathematically in Formula 1. The starting statistical model (Model 0) reflects the main factor used in this experiment (DA) and the Subject random factor $(1|\,Subj.)$. We also tested other statistical models (e.g., models M_1, M_2, M_3, or M_4) that included other fixed factors (Paradigm, Experience, Period, or Sequence), or their combinations, which could have effects on the dependent variables.

$$\begin{array}{ll} (M_0) & DV \sim DA + (1|\,Subj.) \\ (M_1) & DV \sim DA + Paradigm + DA * Paradigm + (1|\,Subj.) \\ (M_2) & DV \sim DA + Paradigm + Experience + (1|\,Subj.) \\ (M_3) & DV \sim DA + Experience + DA * Experience + (1|\,Subj.) \\ (M_4) & DV \sim DA + Sequence + Period + (1|\,Subj.) \end{array} \qquad (1)$$

The statistical model fit of the tested models was evaluated based on goodness of fit measures such as Akaike's information criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC). The model with the smallest AIC or BIC is considered to be the best fitting model [24, 17]. In the analysis, we only considered statistical models verifying the normality of the residuals of the dependent variables. Therefore, to describe the changes in each dependent variable, we selected the statistical model that obtained the smallest AIC or BIC value from the statistical models that satisfied the normality of residuals.

To quantify the differences in the dependent variables due to significant fixed factors, we calculated the Cohen d value [10] between the alternatives of these factors. Cohen d values 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 histograms and box plots to describe the data and the results graphically.

4. Results

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 1 shows the number of subjects of each experiment grouped by experience, "inexperienced" subjects were students, and "experienced" subjects were professionals in the game development industry. The table shows the mean and standard deviation of age, hours per day developing software (Developing time), and hours per day working with models (Modeling time). A 5-point Likert-scale was used for the self-assessment of the subjects' knowledge of programming languages (Programming knowledge) and modeling languages (Modeling Knowledge), which are also shown in Table 1.

Table 1: Results of the demographic questionnaire

	Number of subjects	**Age** $\mu \pm \sigma$	**Developing time**$\pm\sigma$	**Modeling time**$\pm\sigma$	**Programming knowledge**$\pm\sigma$	**Modeling knowledge**$\pm\sigma$
All subjects	79	23.5±6.3	2.5±2.2	0.7±1	3.4±1.2	2.8±1.2
MDD subjects	28	25.8±7.4	3.4±2.6	0.8±1	4±1.1	3±1.3
CDD subjects	51	22.3±5.3	2±1.8	0.6±0.9	3.1±1.1	2.7±1.1
Experienced	26	29.9±7.2	3.8±2.9	1.2±1.2	4.3±0.9	3.5±1.3
MDD Experienced	13	30.7±8.4	4.5±2.7	1.2±1.2	4.6±0.8	3.8±1.4
CDD Experienced	13	29.2±6.1	3.2±2.9	1.2±1.3	4.1±1	3.3±1.3
Inexperienced	53	20.3±1.6	1.9±1.4	0.4±0.6	2.9±1	2.4±0.9
MDD Inexperienced	15	21.5±1.3	2.4±2	0.5±0.6	3.5±1.1	2.3±0.7
CCD Inexperienced	38	19.8±1.5	1.7±1	0.4±0.6	2.7±0.9	2.5±1

4.1. Changes in the dependent variables

The results of the two experiments are shown in Table 2. The first set of rows (ALL) corresponds to the aggregated data of the two experiments; the second and third sets of rows correspond to the data of each experiment individually (CDD and MDD). The table also shows the values for the mean and standard deviation of the dependent variables Correctness, Efficiency, PEOU, PU, and ITU for each of the DAs compared (CaO and SPL) and for each of the alternatives for the fixed factors considered. These include Experience, with two alternatives (Experienced and Inexperienced subjects); Period, with two alternatives (Task 1, and Task 2); and Sequence, whose two alternatives reflect the order in which subjects used the DAs, (G1: CaO-SPL, G2: SPL-CaO). The aggregated data (ALL) should also contain a column with the Paradigm fixed factor and its two alternatives CDD and MDD, but it has been removed to avoid duplicity (those values are also shown under the corresponding rows of CDD and MDD)

Taking into account the aggregated data from the two experiments performed in this study, we can state that there were differences in the means and standard deviations of all of the dependent variables depending on which DA was used to create the boss. If we compare the statistical results obtained in each paradigm separately, we can state that the differences in the dependent variables, depending on which DA was used to create a boss of a video game, were smaller under the MDD paradigm than under the CDD paradigm.

To quantify the differences in the dependent variables, we analyzed the Cohen d values for each of the factors considered in the study. Table 3 shows the Cohen d value of each factor for each dependent variable of the aggregate data of the two experiments performed. Table 4 and Table 5 show the same information but particularized to the data extracted in each of the experiments under the CDD paradigm and the MDD paradigm, respectively. Values indicating high, medium, and small variations due to the factor are shaded in dark grey, grey, and light grey, respectively. Those values corresponding to significant differences according to the hypothesis tests are highlighted in cursive. Positive values indicate differences in favor of the first alternative of the factors, while negative values indicate differences in favor of the second alternative.

According to the Cohen d values of the dependent variables for the DA of the aggregated data (the first column of

Table 2: Values for the mean and standard deviation ($\mu \pm \sigma$) of the dependent variables for the factor Development Approach in each alternative of the fixed factors for data aggregated (ALL), data under the CDD paradigm and data under the MDD paradigm.
Correctness (%) [0-100]; Efficiency (%/min)[0-100]; PEOU,PU,ITU [0-5]

			Development Approach	**Experience**		**Period**		**Sequence**	
				Experienced	Inexperienced	Task 1	Task 2	G1 CaO-SPL	G2 SPL-CaO
ALL	*Correctness*	CaO	53.15±16.98	62.42±10.55	48.52±17.74	53.98±17.87	52.32±16.22	53.98±17.87	52.32±16.22
		SPL	67.75±20.05	75.69±17.93	64.01±20.06	67.91±15.48	67.59±23.98	67.59±23.98	67.91±15.48
	Efficiency	CaO	2.88±1.49	3.37±1.22	2.64±1.57	2.45±1.12	3.31±1.7	2.45±1.12	3.31±1.7
		SPL	3.68±1.69	4.02±1.45	3.53±1.79	3.06±0.97	4.31±2.01	4.31±2.01	3.06±0.97
	PEOU	CaO	3.56±0.85	3.78±0.84	3.45±0.84	3.59±0.88	3.53±0.84	3.59±0.88	3.53±0.84
		SPL	3.97±0.84	4.19±0.95	3.86±0.77	4.18±0.69	3.77±0.94	3.77±0.94	4.18±0.69
	PU	CaO	3.65±0.78	3.88±0.77	3.53±0.77	3.69±0.83	3.6±0.74	3.69±0.83	3.6±0.74
		SPL	3.89±0.76	4.11±0.8	3.78±0.72	4±0.7	3.78±0.8	3.78±0.8	4±0.7
	ITU	CaO	3.25±1.12	3.46±1.12	3.14±1.11	3.3±1.25	3.19±0.97	3.3±1.25	3.19±0.97
		SPL	3.51±1.13	3.69±1.13	3.43±1.13	3.85±0.89	3.19±1.25	3.19±1.25	3.85±0.89
CDD	*Correctness*	CaO	53.87±17.28	64.69±8.14	50.06±18.09	55.93±17.64	51.97±17.07	55.93±17.64	51.97±17.07
		SPL	69.78±11.21	74.83±11.14	68.18±10.9	67.03±11.49	72.75±10.33	72.75±10.33	67.03±11.49
	Efficiency	CaO	2.58±0.92	3.07±0.67	2.41±0.95	2.43±0.97	2.72±0.87	2.43±0.97	2.72±0.87
		SPL	3.9±1.61	4.13±0.94	3.83±1.77	2.95±0.73	4.94±1.66	4.94±1.66	2.95±0.73
	PEOU	CaO	3.57±0.84	3.72±0.71	3.51±0.88	3.51±0.98	3.62±0.7	3.51±0.98	3.62±0.7
		SPL	4.07±0.7	4.45±0.54	3.94±0.7	4.13±0.72	4.01±0.68	4.01±0.68	4.13±0.72
	PU	CaO	3.56±0.8	3.58±0.73	3.56±0.83	3.57±0.89	3.55±0.72	3.57±0.89	3.55±0.72
		SPL	3.99±0.65	4.31±0.48	3.88±0.66	4.02±0.68	3.97±0.63	3.97±0.63	4.02±0.68
	ITU	CaO	3.22±1.11	3±1.17	3.29±1.1	3.18±1.25	3.25±0.99	3.18±1.25	3.25±0.99
		SPL	3.65±1.04	3.73±1.07	3.62±1.04	3.85±0.9	3.44±1.15	3.44±1.15	3.85±0.9
MDD	*Correctness*	CaO	51.87±16.64	60.15±12.43	44.7±16.82	50.87±18.39	53.03±15.02	50.87±18.39	53.03±15.02
		SPL	64.13±29.96	76.48±22.98	53.43±31.84	69.66±21.88	59.34±35.6	59.34±35.6	69.66±21.88
	Efficiency	CaO	3.41±2.09	3.67±1.57	3.19±2.49	2.47±1.37	4.5±2.29	2.47±1.37	4.5±2.29
		SPL	3.29±1.8	3.91±1.83	2.76±1.64	3.27±1.34	3.31±2.16	3.31±2.16	3.27±1.34
	PEOU	CaO	3.54±0.88	3.85±0.97	3.28±0.73	3.72±0.67	3.33±1.07	3.72±0.67	3.33±1.07
		SPL	3.78±1.05	3.92±1.2	3.67±0.93	4.27±0.63	3.37±1.18	3.37±1.18	4.27±0.63
	PU	CaO	3.8±0.74	4.18±0.71	3.5±0.7	3.89±0.69	3.7±0.82	3.89±0.69	3.7±0.82
		SPL	3.7±0.9	3.91±1.01	3.48±0.62	3.97±0.78	3.47±0.96	3.47±0.96	3.97±0.78
	ITU	CaO	3.3±1.14	3.92±0.89	2.77±1.08	.5±1.28	3.08±0.95	3.5±1.28	3.08±0.95
		SPL	3.27±1.26	3.65±1.23	2.93±1.22	3.85±0.9	2.77±1.33	2.77±1.33	3.85±0.9

Table 3), the data under the CDD paradigm (the first column of Table 4), and the data under the MDD paradigm (the first column of Table 5), we can affirm that the differences due to the DA factor were large in Correctness, medium in Efficiency and PEOU, and small in PU and in ITU. In addition, we can state that the differences due to the DA factor were larger under the CDD paradigm than under the MDD paradigm for all of the dependent variables considered in the experiment. Under the CDD paradigm, the effect of DA on Correctness and Efficiency was large and in favor of the SPL approach. In contrast, under the MDD paradigm, the effect of DA on Correctness was medium in favor of SPL and the effect on Efficiency was negligible in favor of CaO. The effect of DA on Satisfaction was also different depending on the paradigm considered: under the CDD paradigm, the effect of DA on Satisfaction was medium-large in favor of SPL; in contrast, under the MDD paradigm, the effect of DA was small for PEOU in favor of SPL, and very small or negligible for PU and ITU in favor of CaO.

The third column of Table 3, and the second columns of Tables 4 and 5 show how the Experience factor had large effects on Correctness, medium effects on Efficiency, and PEOU, and different effects on PU and ITU depending on the paradigm considered. The experienced subjects obtained better results in terms of correctness and efficiency than the inexperienced subjects and were more generous in rating their satisfaction with the approaches used in the experiment. The differences were greater between the experienced and inexperienced subjects under the MDD paradigm than under the CDD paradigm. Under both paradigms, the effect size of Experience for the dependent variables was in the same range for all of the dependent variables except for PU and ITU. Under the CDD paradigm, the experienced subjects rated slightly higher PU than the inexperienced subjects; however, both groups of subjects reported similar values for ITU. In contrast, under the MDD paradigm, the differences between the ratings of experienced and inexperienced subjects were large in favor of the experienced subjects for both PU and ITU.

The Period and Sequence factors also had a different effect on the dependent variables depending on the paradigm under which the subjects performed the experiment (see the last two columns of Tables 3, 4, and 5). Under the CDD paradigm, the Period factor had large effects on Efficiency, but the effects on the other dependent variables were small o negligible. However, under the MDD paradigm, the Period factor had medium-large effects on all of the dependent variables except Correctness. In both experiments, the subjects built bosses of similar correctness in both tasks, but the subjects were more efficient in the second task and rated their satisfaction higher with the approach used on the first task. Under both paradigms, the effect size of Sequence was small for Correctness and medium-large for Efficiency; however, the effects were the opposite depending on the paradigm. Under the CDD paradigm, the subjects in G1 (who started performing the tasks with CaO) obtained better results in Correctness and Efficiency, while under the MDD paradigm, the subjects in G2 (who started performing the tasks with SPL) obtained better results in Correctness and Efficiency. Under both paradigms, the subjects who started with SPL reported being more satisfied than the subjects who started with CaO.

Table 3: Cohen d values for the independent variables for each fixed factor.

ALL	**Development Approach** (CaO/SPL)	**Paradigm** (CDD/MDD)	**Experience** (Experienced/ Inexperienced)	**Period** (Task 1/ Task 2)	**Sequence** (G1(CaO-SPL)/ /G2(SPL-CaO))
Correctness	*-0,786*	0,181	*0,687*	0,049	0,034
Efficiency	*-0,503*	-0,065	*0,387*	-0,682	0,119
PEOU	*-0,485*	0,172	*0,378*	*0,266*	-0,2
PU	-0,317	0,034	*0,432*	0,2	-0,086
ITU	-0,237	0,127	*0,262*	*0,341*	-0,245

Table 4: Cohen d values for the independent variables for each fixed factor under the CDD paradigm.

CDD	**Development Approach** (CaO/SPL)	**Experience** (Experienced/ Inexperienced)	**Period** (Task 1/ Task 2)	**Sequence** (G1(CaO-SPL)/ /G2(SPL-CaO))
Correctness	*-1.092*	*0.713*	-0.014	0.294
Efficiency	*-1.008*	0.344	-0.794	*0.599*
PEOU	*-0.651*	0.462	0.016	-0.146
PU	*-0.594*	0.301	0.058	-0.024
ITU	*-0.4*	-0.079	0.161	*-0.217*

Table 5: Cohen d values for the independent variables for each fixed factor under the MDD paradigm

MDD	**Development Approach** (CaO/SPL)	**Experience** (Experienced/ Inexperienced)	**Period** (Task 1/ Task 2)	**Sequence** (G1(CaO-SPL)/ /G2(SPL-CaO))
Correctness	*-0.506*	*0.843*	0.127	-0.254
Efficiency	0.062	0.433	*-0.544*	-0.529
PEOU	-0.251	0.428	*0.675*	-0.265
PU	0.119	*0.684*	0.441	-0.186
ITU	0.030	*0.853*	*0.658*	-0.279

4.2. Answers to the Research Questions

Table 6 shows the results of the Type III fixed effects test applied on the selected statistical models for each of the dependent variables and for each fixed factor considering the aggregated data. Tables 7 and 8 show the same information as Table 6 but related only to the data extracted during each of the experiments, under the CDD and the MDD paradigm, respectively. The values indicating significant differences are shaded in grey. The statistical model selected for each variable consists of the factors for which statistical significance is shown in Tables 6, 7, and 8. If NA appears, it is because the factor in that column is not part of the fixed factors considered in the statistical model selected for the variable in that row. For example, the model selected for Correctness contains all of the fixed factors that appear as headings in the table and the random factor. Its mathematical representation would be $DV \sim DA + Exp + Period + Sequence + DA * Exp + Exp * Period + Exp * Sequence + (1|\, Subj.)$.

The factors and combinations of factors that are part of the selected statistical models explain the changes in the dependent variables; however, according to the test results of the hypotheses, not all the changes in the dependent variables due to these factors were significant. Statistical models such as those in Formula 1 and other LMM tested did not verify the normality of the residuals or obtain higher values for the AIC and BIC fit statistics than those obtained by the selected models. With the dataset from the experiment under the CDD paradigm, we obtained normally distributed residuals for Efficiency and PEOU by using continuous transformations. We used square root transformation for Efficiency, and we used the transformation of squaring for PEOU.

4.2.1. Impact of the Development Approach on Correctness

For Correctness, the DA factor obtained a p-value of less than 0.05 regardless of the paradigm considered. Therefore, we reject the first null hypothesis, and the answer to **RQ1** is affirmative for both paradigms: The DA used for creating a boss in a video game had a significant impact on Correctness regardless of whether the comparison was made under CDD paradigm or under MDD paradigm. In addition, the combination of DA and Paradigm was considered to be statistically significant to explain the changes in Correctness. The bosses made using the SPL were more correct than those made using CaO regardless of whether the comparison was performed under the CDD paradigm or under the MDD paradigm. In addition, the benefits for Correctness of using SPL over using CaO were larger under the CDD paradigm than under the MDD paradigm. The results of the hypothesis tests confirm the statistical significance of the differences observed in the mean values of Correctness for the alternatives of the combination of DA and Paradigm: CDD-CaO (53.87 ± 17.28), CDD-SPL (69.78 ± 11.21), MDD-CaO (51.87 ± 16.64), and MDD-SPL (64.13 ± 29.96).

The effect size of a factor measure through the Cohen d value is related to the percentage of non-overlap between the distributions of the dependent variables for each alternative of the factor. Higher effect sizes correspond with greater percentages of non-overlap and larger differences. The histograms in Figure 5 illustrate the differences in Correctness depending on the DA factor and the combination DA and Paradigm. The first column considers the aggregated data of the two experiments performed, the second column shows the results in Correctness un-

Table 6: Results of the Type III test of fixed effects for each dependent variable and factor. NA=Not Applicable

ALL	**Development Approach (DA)**	**Paradigm**	**Experience (Exp)**	**Period**	**Sequence**	**DA*Paradigm**	**Paradigm*Exp**	**Paradigm*Period**	**Paradigm*Sequence**
Correctness	F=33.284; p=<.001	F=2.963; p=0.089	F=19.644; p=<.001	F=0.31; p=0.579	F=0.09; p=0.765	F=0.577; p=0.45	F=1.03; p=0.314	F=1.146; p=0.288	F=2.704; p=0.104
Efficiency	F=7.664; p=0.007	F=0.017; p=0.895	F=4.163; p=0.045	NA	F=0.83; p=0.366	F=11.167; p=0.001	F=0.279; p=0.599	NA	F=10.825; p=0.002
PEOU	F=16.765; p=<.001	F=2.367; p=0.128	F=5.54; p=0.021	F=9.843; p=0.002	NA	NA	NA	F=9.118; p=0.003	NA
PU	F=2.127; p=0.149	F=0.21; p=0.648	F=7.293; p=0.009	NA	NA	F=5.376; p=0.023	F=1.128; p=0.292	NA	NA
ITU	F=1.607; p=0.209	F=0.193; p=0.662	F=5.217; p=0.025	F=6.833; p=0.011	F=2.439; p=0.123	F=1.358; p=0.248	F=6.956; p=0.01	F=2.764; p=0.101	NA

Table 7: Results of the Type III test of fixed effects for each variable and factor under the CDD paradigm. NA=Not Applicable

CDD	**Development Approach (DA)**	**Experience (Exp)**	**Period**	**Sequence**	**DA*Experience**	**Exp*Sequence**	**Exp*Period**
Correctness	F=31.717; p=<.001	F=8.651; p=0.005	F=2.251; p=0.14	F=1.128; p=0.294	F=2.521; p=0.119	F=0.031; p=0.861	F=3.134; p=0.083
Efficiency	F=42.527; p=<.001	F=1.037; p=0.314	NA	F=21.086; p=<.001	F=0.904; p=0.347	NA	NA
PEOU	F=20.293; p=<.001	F=3.545; p=0.066	F=0.223; p=0.639	F=1.304; p=0.259	F=2.322; p=0.134	F=1.339; p=0.253	F=0.005; p=0.941
PU	F=13.719; p=0.001	F=1.482; p=0.229	NA	NA	F=1.972; p=0.166	NA	NA
ITU	F=5.007; p=0.03	F=0.043; p=0.836	NA	F=3.692; p=0.061	F=0.72; p=0.4	F=3.565; p=0.065	NA

Table 8: Results of the Type III test of fixed effects for each variable and factor under the MDD paradigm. NA=Not Applicable

MDD	**Development Approach (DA)**	**Experience (Exp)**	**Period**	**Sequence**	**DA*Experience**	**DA*Period**	**Exp*Period**	**Exp*Sequence**
Correctness	F=5.448; p=0.028	F=8.531; p=0.007	F=0.574; p=0.456	F=0.737; p=0.399	F=0.501; p=0.486	NA	F=0.031; p=0.863	F=4.34; p=0.048
Efficiency	F=0.177; p=0.678	F=1.932; p=0.177	F=6.379; p=0.019	F=2.908; p=0.101	F=0.754; p=0.394	NA	F=0.37; p=0.549	F=0.632; p=0.435
PEOU	F=1.735; p=0.192	F=2.257; p=0.145	F=9.111; p=0.004	NA	F=0.572; p=0.452	F=0.885; p=0.356	F=0.216; p=0.643	NA
PU	F=0.17; p=0.684	F=6.139; p=0.02	F=3.026; p=0.095	NA	F=0.565; p=0.46	F=0.514; p=0.48	F=0.111; p=0.742	NA
ITU	F=0; p=0.993	F=11.899; p=0.002	F=6.293; p=0.019	NA	F=0.515; p=0.48	F=1.493; p=0.233	F=0; p=0.986	NA

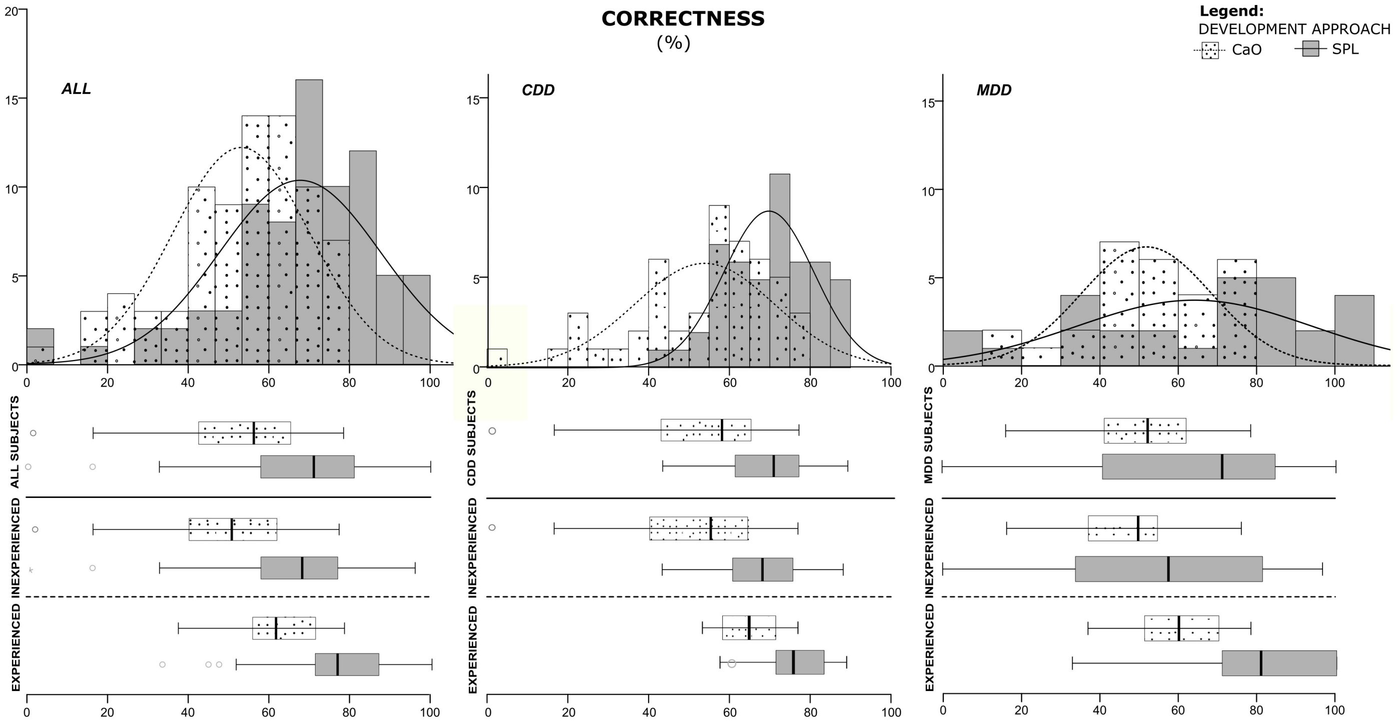


Figure 5: Histograms with normal distributions and box plots for Correctness

der the CDD paradigm, and the third column shows the results for Correctness under the MDD paradigm. The non-overlapping parts have a single pattern (either dotted or shaded), while the overlapping parts have both patterns (dotted and shaded). In the first histogram, the non-overlapping parts for Correctness are around 30%, which corresponds to a large effect size and to a Cohen *d* value of around 0.8. Under the CDD paradigm, there are more than 40% of non-overlapping parts, which corresponds to a very large effect size and to a Cohen *d* value higher than 1.0. Under the MDD paradigm, they are around 20%, which corresponds to a medium effect size and to a Cohen *d* value of around 0.5.

The bottom part of Figure 5 shows nine pairs of box plots, arranged in three rows and columns, illustrating the differences in Correctness due to DA. The first row of box plots corresponds to all of the subjects. The second row corresponds to the inexperienced subjects, and the third row corresponds to the experienced subjects. The median of the values of Correctness is higher for SPL than for CaO in all of the box plots and is also higher for experienced subjects than for inexperienced ones. Both the experienced and inexperienced subjects developed more correct bosses when using SPL instead of CaO regardless of the paradigm considered. However, the bosses created by the experienced subjects were more correct than those created by the inexperienced subjects. The Experience factor was considered to be statistically significant to explain the changes in Correctness under both paradigms. However, the changes due to the combination of DA and Experience (DA*Exp) and the combination of Paradigm and Experience (Paradigm*Exp) were not considered to be statistically significant, indicating that the differences in Correctness between the experienced and inexperienced subjects did not depend on either the DA or the Paradigm.

4.2.2. Impact of the Development Approach on Efficiency

For Efficiency, the DA factor obtained a p-value of less than 0.05 for the aggregated data under the CDD paradigm, but greater than 0.05 under the MDD paradigm. Therefore, we cannot reject the second null hypothesis under the MDD paradigm and the answer to the research question **RQ2** is different depending on the paradigm under consideration. Under the CDD paradigm, the DA used had a significant impact on Efficiency and its effect size was large in favor of SPL. In contrast, under the MDD paradigm, the DA used did not have a significant impact on Efficiency. Under the CDD paradigm, the subjects spent less time creating the boss when using the SPL (19.69 ± 5.62min.) than when using CaO (21.65 ± 4.93 min.). Under MDD paradigm, the subjects spent more time creating the boss when using the SPL (20.6 ± 5.72 min.) than when using CaO (18.43 ± 7.33 min.), which counteracted the positive effect of the use of SPL on Correctness. In addition, the combination of DA and Paradigm was statistically significant to explain the changes in Efficiency. The mean values in Efficiency for the alternatives of these factors indicate that the differences in Efficiency between using the SPL or using CaO were large in favour of the SPL under the CDD paradigm, but were negligible under the MDD paradigm. The results of the hypothesis tests confirm the statistical significance of these differences.

The histograms in Figure 6 illustrate the differences in Efficiency depending on the DA factor and the combination DA and Paradigm. The first column considers the aggregated data of the two experiments performed. The second column shows the results in Efficiency under the CDD paradigm, and the third column shows the results for Efficiency under the MDD paradigm. In the first histogram, the non-overlapping parts for Efficiency are around 20%, which corresponds to a medium effect size. Under the CDD paradigm, they are more than 40%, which corresponds to a very large effect size, and, under the MDD paradigm, they are less than 3%, which corresponds to a negligible effect size.

The factor Experience was considered to be statistically significant to explain the changes in Efficiency. Its effect was large in favor of the experienced subjects. However, the changes due to the combination of DA and Experience (DA*Exp) and the combination of Paradigm and Experience (Paradigm*Exp) on Efficiency were not significant. This indicates that the experienced subjects were more efficient than the inexperienced subjects regardless of the Development approach used (CaO or SPL) or the development paradigm (MDD or CDD). The box plots of Figure 6 show the differences in Efficiency due to DA for all of the subjects (the first column), for the subjects participating in the experiment under CDD paradigm (second column), and for the subjects in the experiment under the MDD paradigm (third column). The median of Correctness is higher for SPL than for CaO in all of the box plots. The box plots of Figure 6 corresponding to the experienced and inexperienced subjects (the last two rows) show the differences in Correctness due to the combination of Experience and DA.

4.2.3. Impact of the Development Approach on Satisfaction

For the dependent variables related to satisfaction, the DA factor obtained p-values of less than 0.05 under the CDD paradigm but greater than 0.05 under the MDD paradigm. Therefore, we can reject the null hypotheses related to satisfaction only under the CDD paradigm and the answer to research question **RQ3** is different depending on the paradigm under consideration. Under the CDD paradigm, the DA factor had a significant impact in all of the variables related to satisfaction: PEOU, PU, and ITU. In contrast, under the MDD paradigm, DA did not have a significant impact on the satisfaction of the subjects when creating a video game boss. The combination of the DA and Paradigm factors was statistically significant for PU. The subjects perceived that the SPL approach was more useful than the CaO approach, especially under the CDD paradigm.

The Experience factor had significant and large effects on satisfaction only under the MDD paradigm: the experienced subjects scored higher than the inexperienced subjects on PU and ITU for either of the two approaches used in the experiment. However, under the CDD paradigm, the differences in PU and ITU between the experienced and inexperienced subjects were not statistically significant. Under both paradigms, the changes in PEOU due to Experience were not statistically significant.

Figure 7 shows fifteen pairs of box plots arranged in five rows and three columns, illustrating the differences in PEOU (the first column), in PU (the second column), and in ITU (the third column) due to DA. The first row of box plots shows the differences for all subjects from both experiments. The second and third rows of box plots

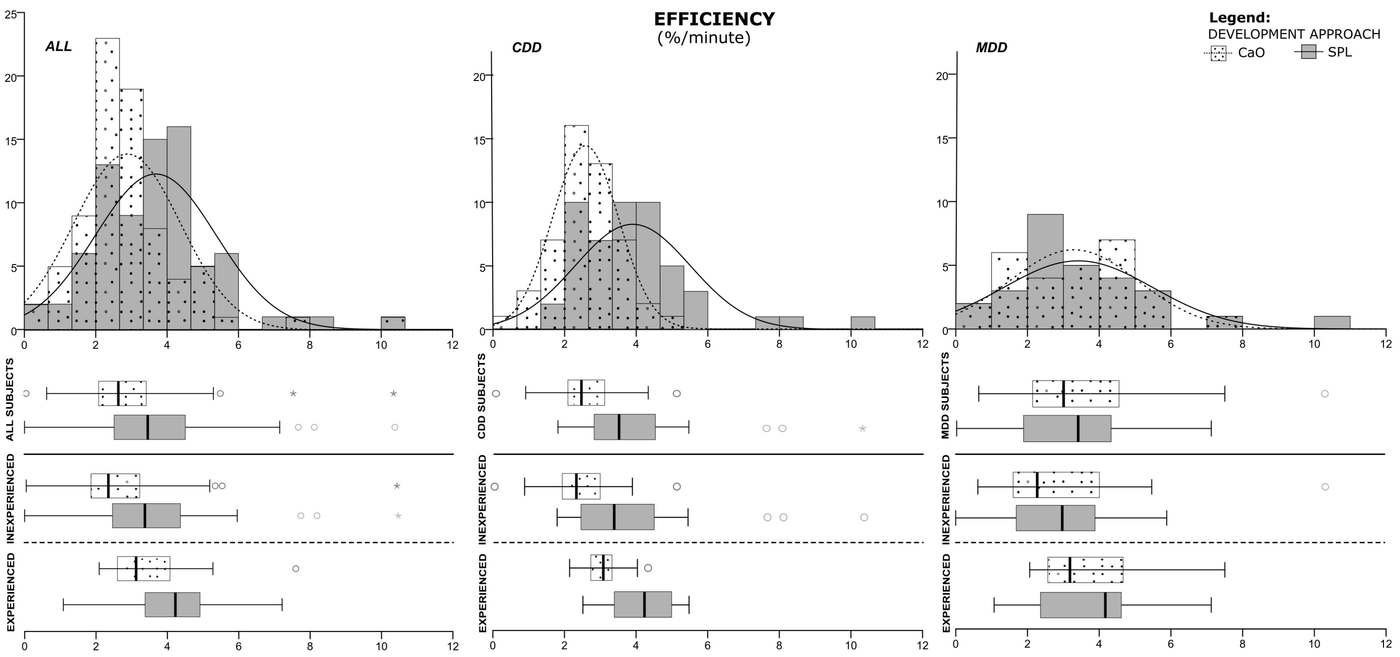


Figure 6: Histograms with normal distributions and box plots for Efficiency

show the differences due to the combination of DA and Paradigm. The fourth and fifth rows of box plots show the differences due to the combination of DA and Experience. The values for the median of all of the variables related to satisfaction are higher for SPL than for CaO in all of the box plots, and they are also higher for the experienced subjects than for the inexperienced ones. However, there is hardly any variation between the box plots of the different paradigms for PEOU, while the box plots for PU and ITU show that the differences in favor of SPL are larger under the MDD paradigm than under the CCD paradigm.

4.2.4. Impact of Paradigms on the Differences due to the Development Approach

From the above results, we can answer question **RQ4**. The paradigm under which video game elements were created significantly affected the comparison between the development approaches in terms of Correctness and Efficiency. The differences in favor of using an SPL instead of CaO were greater under the CDD paradigm than under the MDD paradigm, especially in Efficiency. Under CDD there were very large differences in Efficiency in favor of using an SPL, but, under the MDD paradigm, the use of an SPL did not provide significant benefits in Efficiency over the use of CaO. The differences in Satisfaction between the use of SPL and CaO in the two paradigms are not as clear. Although the Satisfaction values in favor of SPL were higher under the CDD paradigm, the differences between the paradigms were only significant in PU, and the differences in all variables related to Satisfaction (PEOU, PU and ITU) in favor of SPL were lower than those obtained in Efficiency or Correction in either of the two paradigms considered.

5. Discussion

As reported over the last two decades by the Software Engineering Institute of the Carnegie Mellon University[3], SPLs can multiply productivity by a factor of 10, reduce costs up to 60%, reduce labor needs up to 87%, and reduce the time-to-market of new software variants up to 98%. These benefits have been used repeatedly for years to show the attractiveness of SPLs for CSE.

However, our results in the video game domain reveal that efficiency is not always the key aspect that provides SPLs with an advantage over CaO. Existing differences between CSE and GSE (e.g., working on different kinds of artifacts or how they perceive the development process of their projects) may mean that other aspects and uses of SPLs are more relevant for GSE. To better understand the results of the experiments, we carried out focus groups with the subjects.

In addition, the benefits in efficiency of using the SPL instead of CaO were not significant for the subjects in the MDD paradigm, but, under the CDD paradigm, most of the subjects acknowledged that the SPL did accelerate video game development (also shown by our results). When asked about it, the subjects under the MDD paradigm claimed that there was not much difference in working with one model when using CaO (SDML) or the other models when using SPL (initial model + variability models); they were similar for the subjects. However, the subjects under the CDD paradigm reported that when using the CaO approach, they had to think in terms of C++,

[3]Available through their online library collection: `https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=513819`

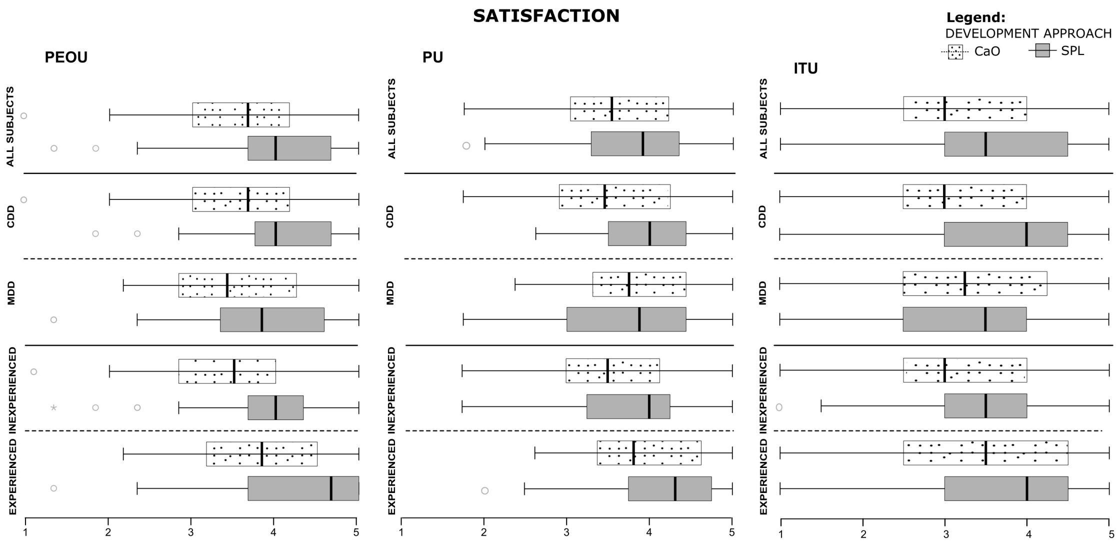


Figure 7: Box plots for the dependent variables related to Satisfaction: PEOU, PU, and ITU

but when working under the MDD paradigm, they were just thinking in terms of the feature model, abstracting away the C++ details. The change between working with SPL or CaO seemed bigger under the CDD paradigm, as they effectively abstracted from code details to focus on the feature model. This may also be augmented by the fact that the SPL used in MDD followed a compositional approach, while the SPL used in CDD used an annotative approach, facilitating the derivation process.

Additionally, most of the subjects observed benefits in SPL for video games, aside from the task requested in the experiments. Specifically, most of the subjects mentioned that the SPL was very relevant for creating new content for video games. As they explained, creating content is critical for video game development. Video game worlds may need a large amount of content to populate them (e.g., non-player characters or items such as weapons or power-ups). Furthermore, there is a trend in the video game industry toward the Games as a Service (GaaS) business model [51]. Once the video game is launched, it is kept alive through updates and additional Downloadable Content (also called DLCs). This new content needs to be cohesive with the already released game.

The subjects reported that thinking in terms of SPL features helped them think of new content for the video game. The subjects positively valued that they could obtain new content by combining the features following the rules of the variability specification. This new content was different but similar to what already existed in the video game, and the subjects affirmed that this content was relevant to developing video games. The subjects imagine that this new content derived from assembling features can be useful as variants of what was already used in the video game in order to avoid repetition. It could also be used to fill in secondary parts of the game that are more sparsely populated or even as a source of inspiration to create completely new content.

A group of the subjects reported that the SPL may be relevant in helping to balance the difficulty of video games. Balancing the difficulty of video games is one of the main problems of video game development. The video game cannot be too difficult because the players would get frustrated, but it cannot be too easy because the players would lose interest. The subjects thought about how the features could be augmented to include the idea of difficulty in order to derive variants of the same content with different degrees of difficulty.

In the video game research community, content generation for video games is a hot topic. Many surveys [23, 34, 47, 49] talk about how to (completely or partially) automate content generation. Specifically, the automation of content generation is known as Procedural Content Generation (PCG) [3] and uses the most popular Computational Intelligence techniques today (such as machine learning and search-based). However, the SPLs do not appear in any of those surveys. This suggests that the use of SPLs to generate video game content has been neglected by the video game research community and may be an opportunity to explore in the future.

Moreover, we suggest that SPLs need not be viewed solely as an alternative to the current PCG techniques; SPLs can be an ally when combined with PCG techniques. One of the subjects highlighted that the strength of SPL compared to CaO was that only 10 decisions were needed in SPL, while he had to make between 50 and 100 decisions in CaO. This reduced search space of SPLs may be more favorable for the machine learning techniques used in PCG to extract patterns or for the search-based techniques used in PCG to explore the search space.

The experienced subjects acknowledge the benefits of SPLs over CaO under the CDD paradigm. They say SPLs are easier, more accessible, less error-prone, and clearer to understand. However, they sense that they lose expressiveness and control over the software they create. As we have seen in Section 4, there are significant differences in favor of SPLs for all of the variables, with the exception of *Intent To Use*. Even though results are favorable toward SPLs, the experienced subjects value source code control of CaO over the many benefits that SPLs offer. Extending the annotated source code was out of the scope of the experiment, but this lack of control would be mitigated in a real scenario, where the developers could also modify the annotated code if needed.

Game development teams are multidisciplinary, and software engineers work together with plenty of non-technical roles (artists, designers, musicians, etc.) that carry the creative part of video games. We claim that the accessibility of SPLs can bring software creation to these roles in a game development team. The engineers focus on building the foundations of the video game with an SPL that comprises the requirements needed, and the other team members can focus on creating content for the game that is new and cohesive with the rest of the content.

One caveat of GSE is that, in general, games have shorter development times than applications in CSE. There is a compromise to be made when building an SPL system to accelerate development in later development stages. Long developments such as those described in a GaaS context can benefit more from SPLs than short-lifetime video games.

We have identified that the industrial-scale artifact of this work is a game that has already been published. As future work, further research with other video games still in development can bring new perspectives to using SPLs in the video game domain. Finally, one key aspect of video games is that there are many different platforms on which to publish the games. Computers, consoles, and mobiles have plenty of differences that developers need to address when releasing the same game on these platforms. We encourage researchers to explore the benefits of SPLs in this stage of video game development where software reuse and variability are the main artifacts.

6. Threats to validity

To describe the threats to validity of our work, we have used the classification of Wohlin et al. [54]:

Conclusion validity is achieved when there is a statistical relationship (with a certain significance) between the treatment and the results. The *low statistical power* was minimized because the confidence interval was 95%. To minimize the *fishing and the error rate* threat, the statistical analysis was done by a researcher who did not participate in the task design or in the correction process. The *Reliability of measures* threat was mitigated because the objective measurements were obtained from the digital artifacts generated by the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the procedure was identical in each of the sessions in which the experiment was performed.

Internal validity is achieved when the observed relationships between treatment and outcome are causal relationships and these relationships are not the result of a factor over which we have no control or we have not measured. To avoid the *instrumentation* threat, we conducted a pilot study to verify the design and the instrumentation. The *maturation* threat affected the experiment. Even though the tasks were designed with similar complexity, the effect of period was significant for Efficiency, PEOU, and ITU. The subjects were more efficient in the second task, while PEOU and ITU were better valued in the first task. We minimized this threat by using a crossover design so that the effect affected the evaluated approaches equally. The *selection* threat also affected the experiment because of the voluntary nature of participation. To avoid student demotivation, we selected students from courses whose contents fit the design of the experiment. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of modeling language knowledge and different levels of knowledge of the video game domain. To mitigate this threat, the treatment was applied randomly and the effects of the sequence factor have been included in the statistical analysis. On the other hand, the number of subjects participating in the experiment under the CDD paradigm was higher than the number of subjects participating under the MDD paradigm, which makes the CDD paradigm more represented in the results associated with the aggregated data. Similarly, inexperienced subjects are more represented in the overall results than experienced subjects.

Construct validity is achieved when the measures actually represent what is being investigated based on the research questions. *Mono-method bias* occurs due to the use of a single type of measure [39]. To mitigate this threat to the correctness and efficiency measurements, we mechanized these measurements as much as possible by means of correction templates. We mitigated the threat to satisfaction measurements by using a widely applied model (TAM) [13]. 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 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 of their abilities, and in the case of students, that neither their participation nor their results would affect their grades in the course. *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 extracted from a commercial video game and were designed by the same expert and researcher with similar difficulty for all of the tasks. These people did not participate in the experiment execution.

External validity is achieved when the results can be generalized outside the settings of the experiment. 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. The participation of students rather than software engineers is not a major issue as long as the research questions are not specifically focused on experienced developers [28], as is the case in this experiment. In addition, the experience factor has been taken into account in the data analysis. The *interaction of setting and treatment threat* affected the experiment. The focus group revealed that, for both development approaches and paradigms, the subjects spent a lot of time evaluating if the boss being created was the boss they wanted to develop. The subjects stated that they spent a lot of time testing the video game in real time to understand what they had to develop. This should not be a problem in a real development scenario because it is assumed that the game engineers understand the game they are creating. The *domain* threat occurred because the experiment was conducted in a specific domain (video game) and for a very specific type of task, i.e., to develop a video game boss from Kromaia. We think that the generalization of findings should be undertaken with caution. Other experiments in different games should be performed to validate our findings.

Regarding the generalization of findings, it is important to note the replication limitation of this study. Although we share all the data related to the study results, methodology, and statistical analysis, some of the artifacts related to the infrastructure cannot be disclosed (e.g., game project, DSL, SPL). To mitigate this threat, the experiment was performed using a subset of these systems. We used a subset of the Kormaia content, a subset of the DSL, and a subset of the SPL for the tasks to be understandable and completed in a reasonable amount of time (2 hours). Using these subsets allows both experiments to be performed likewise, using pen and paper, without the need for the complete software infrastructure. However, to evaluate the tasks is needed an experienced researcher in the domain.

All the artifacts needed to perform both experiments are available, along with the complete data set of the results and statistical analysis, in the attached link of Section 3.7.

7. Related Work

This section presents the related works. It is divided into two subsections taking into account the topics covered in this paper: SPLs applied to the video game domain, and SPL and CaO evaluations or studies.

7.1. SPLs applied to the video game domain

Some approaches are focused on managing the inherent variability of video games. Boaventura and Sarinho [7] present a collection of game assets to create small video games called Minimal Engine for Digital Games (MEnDiGa). MEnDiGa extends a previous work called FEnDiGa [43], a product line platform that is able to integrate and adapt represented game features in different types of available game engines. In MEnDiGa, they develop logic features and modules to represent, interpret, and adapt game features for functionality in multiple game platforms. Their results show that the core of game elements developed with MEnDiGa can be independent, reusable, and done in a large-scale way. Castro and Werner [9] present a prototype of a game that was developed by applying a dynamic SPL to generate game modifications systematically. The prototype created demonstrates the possibility of automating the process, having a product line where the original game is the core of the game's functionalities.

Some other research efforts are focused on the creation of SPLs applying re-engineering. Lima et al. [32, 31] present two works about Product Line Architecture (PLA) recovery. They applied their previous proposed approach and guidelines [33] to create the PLA of the Apo-Games project [29]. In addition, they developed an automatic approach for the identification of the minimum subset of cross-product architectural information for an effective PLA recovery. Moreira et al. [38] provide and analyze empirical data of the extraction processes of two open-source case studies, ArgoUML and Phaser. Although the results show that there is a great difference between the re-engineering process of ArgoUML-SPL and Phaser, common problems were found in both cases, which were related to a lack of tools that led to incomplete and inconsistent feature extractions, complexity in managing feature dependencies when using compositional approach, and issues of not having a variability model for dealing with feature constraints.

Similarly, Martinez et al. [35] report an experience concerning the creation of an SPL by re-engineering system variants implemented around an educational game called Robocode. They discuss their results from different perspectives such as educational value, the extractive process, and the analysis of the time and effort required. Debbiche et al. [14] analyze the Apo-Games to identify reusable code or artifacts. They analyzed 5 Java games and migrated 3 of these into a composition-based SPL implemented with *FeatureHouse* [2]. They recommend making sure that the SPL is always testable to ensure the correct transformation to code, and they also recommend incrementally adopting new features to facilitate the extraction.

The above works apply SPL approaches in the video game domain. However, these works do not evaluate whether the benefits produced by the use of SPLs in CSE apply in GSE. In our work, we evaluate the SPL with a commercial video game and with subjects that one linked to the development of video games.

7.2. SPL and CaO evaluations or studies

Some works focus on comparing different SPL techniques. Constantino et al. [11] compare two SPL tools taking into account ease of use, strengths, and weaknesses. Their results show that the main problem for both tools is the lack of user guides; however, the automated analysis and the feature model editor get very good ratings. Dermerval et al. [15] compared two approaches for feature modeling based on ontologies. Their results suggest that the reasoning capabilities of the ontologies can effectively support product reconfiguration in the context of Dynamics Software Product Lines (DSPLs).

In the context of MDD, Gonzalez-Huerta et al. [22] carried out a family of four experiments to evaluate an MDD method for developing SPLs. They used effectiveness, efficiency, PEOU, PU, and ITU in their comparison. Their experiments show that the method based on MDD obtains the best results.

Some of the research efforts related to CaO focus on features. Ghabach et al. [21] proposed an approach to support the derivation of new product variants based on CaO. Their evaluation of a case study shows that the provided support can considerably reduce the amount of time and effort required to achieve a product derivation. Similarly, Krüger et al. [30] proposed a semi-automatic process to identify and map features between legacy systems. Their results indicate that the process can be used to enable traceability, prepare refactorings, and extract SPLs. Kehrer et al. [27] presented a project called VariantSync to bridge the gap between CaO and SPLs by combining the minimal overhead and flexibility of CaO with the systematic handling of variability in SPLs. They believe that VarianSync has great potential to change the way practitioners develop multi-variant software systems.

After analyzing the above works, we realized that there is a lack of empirical studies comparing SPLs and CaO. Along with the previous paper [50] that this work extends, we were able to find only one paper that performed this comparison [18]. The authors compare the performances of software engineers in the software product development process using SPL and CaO in an industrial context. Their results achieved better values for effectiveness, efficiency, and satisfaction with the SPL approach. In contrast to our work, they focused on CSE and did not address GSE.

8. Conclusion

In this paper, we have presented two experiments that evaluate whether the adoption of an SPL in GSE can generate similar benefits as in CSE [18] in terms of correctness, efficiency, and satisfaction. In our experiments, we compared two development approaches, CaO and SPL. Each experiment was conducted under the MDD and CDD paradigms, respectively. A total of 79 subjects participated in the experiments (28 in MDD and 51 in CDD) by developing the final bosses of Kromaia, a commercial video game.

The results show that the bosses developed using the SPL approach were more correct than the bosses developed with CaO under both paradigms. However, there were no significant changes in efficiency when working under the MDD paradigm, while the changes were large and significant when working under the CDD paradigm. Similarly, changes in satisfaction favored SPL rather than CaO and were larger when working under the CDD paradigm than when working under the MDD paradigm. It turns out that the use of SPLs for GSE has a greater impact under the CDD paradigm (where the differences are bigger) than under the MDD paradigm. The subjects of the experiment reported that the change in the way of working with SPL under the CDD paradigm was a real benefit. They could program without code knowledge, focusing only on the feature model and leaving the actual code in the background to the experts.

The results of the previous work [50] revealed that SPLs in GSE can be relevant in generating new video game content (a hot topic in the video game research community) and in balancing the video game difficulty (a seminal problem of video games). This extension supports the relevance of SPLs in GSE, where the differences in favor of SPLs over CaO are greater in a CDD paradigm. Overall, SPLs perform better in terms of correctness, efficiency, and satisfaction under a CDD paradigm. However, the results show that the satisfaction manifested by the subjects with the use of an SPL instead of CaO is not proportional to the benefits that an SPL may have in correction or efficiency, especially in terms of intention to use. We see that developers are still reluctant to use SPLs rather than CaO because of the sense of loss of control over the software they create, even though they perform better when using SPLs.

While SPLs performed better in every aspect than CaO, the SPL approach in GSE did not have the same effects that have made SPLs attractive to CSE. Other applications of SPLs in different games, of different genres, and with different content should be performed to validate the generalization of findings.

There are new trends, such as the GaaS business model, and the particularities of the video game domain that differ from CSE, such as multidisciplinary teams, content creation, difficulty balancing, or the large range of different platforms. These trends and particularities bring novel uses for SPLs within the fertile domain of video games. Therefore, we hope that this work encourages further research of SPLs in GSE.

Acknowledgments

This work has been partially supported by Iniciando Lineas de Producto Software Mediante Busquedas Interactivas Dirigidas por Modelos. Reference: PID2021-128695OB-I00, by Centro para el Desarrollo Tecnológico y la Innovación (CDTI). Reference: SAV-20221011, and in part by the Gobierno de Aragón (Spain) (Research Group S05 20D).

References

[1] Ampatzoglou, A., Stamelos, I., 2010. Software engineering research for computer games: A systematic review. Information and Software Technology 52, 888 – 901. doi:`https://doi.org/10.1016/j.infsof.2010.05.004`.

[2] Apel, S., Kästner, C., Lengauer, C., 2013. Language-independent and automated software composition: The featurehouse experience. IEEE Transactions on Software Engineering 39, 63–79. doi:`10.1109/TSE.2011.120`.

[3] Barriga, N.A., 2019. A short introduction to procedural content generation algorithms for videogames. International Journal on Artificial Intelligence Tools 28, 1930001. doi:`10.1142/S0218213019300011`.

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

[5] Blasco, D., Cetina, C., Pastor, O., 2020. A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study. Inf. Softw. Technol. 119. doi:`10.1016/j.infsof.2019.106235`.

[6] Blasco, D., Font, J., Zamorano, M., Cetina, C., 2021. An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering. Journal of Systems and Software 171, 110804.

[7] Boaventura, F.M.B., Sarinho, V.T., 2017. Mendiga: A minimal engine for digital games. Int. J. Comput. Games Technol. 2017, 9626710:1–9626710:13. doi:`10.1155/2017/9626710`.

[8] Borowa, K., Zalewski, A., Saczko, A., 2021. Living with technical debt—a perspective from the video game industry. IEEE Software 38, 65–70.

[9] Castro, D., Werner, C., 2021. Rebuilding games at runtime, IEEE. pp. 73–77. doi:`10.1109/ASEW52652.2021.00025`.

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

[11] Constantino, K., Pereira, J.A., Padilha, J., Vasconcelos, P., Figueiredo, E., 2016. An empirical study of two software product line tools, SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT. p. 164–171. doi:`10.5220/0005829801640171`.

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

[13] Davis, F.D., 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Q. 13, 319–340.

[14] Debbiche, J., Lignell, O., Krüger, J., Berger, T., 2019. Migrating java-based apo-games into a composition-based software product line, ACM. pp. 18:1–18:5. doi:`10.1145/3336294.3342361`.

[15] Dermeval, D., Tenório, T., Bittencourt, I.I., Silva, A., Isotani, S., Ribeiro, M., 2015. Ontology-based feature modeling: An empirical study in changing scenarios. Expert Systems with Applications 42, 4950–4964. doi:`https://doi.org/10.1016/j.eswa.2015.02.020`.

[16] Domingo, Á., Echeverría, J., Pastor, O., Cetina, C., 2020. Evaluating the benefits of model-driven development - empirical evaluation paper, Springer. pp. 353–367. doi:`10.1007/978-3-030-49435-3\_22`.

[17] Domingo, Á., Echeverría, J., Pastor, Ó., Cetina, C., 2021. Comparing uml-based and dsl-based modeling from subjective and objective perspectives, Springer. pp. 483–498.

[18] Echeverría, J., Pérez, F., Panach, J.I., Cetina, C., 2021. An empirical study of performance using clone & own and software product lines in an industrial context. Inf. Softw. Technol. 130, 106444. doi:`10.1016/j.infsof.2020.106444`.

[19] Fischer, S., Linsbauer, L., Lopez-Herrejon, R.E., Egyed, A., 2015. The ecco tool: Extraction and composition for clone-and-own, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE. pp. 665–668.

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

[21] Ghabach, E., Blay-Fornarino, M., Khoury, F.E., Baz, B., 2018. Clone-and-Own Software Product Derivation Based on Developer Preferences and Cost Estimation, Nantes, France.

[22] Gonzalez-Huerta, J., Insfran, E., Abrahão, S., Scanniello, G., 2015. Validating a model-driven software architecture evaluation and improvement method: A family of experiments. Information and Software Technology 57, 405–429. doi:`https://doi.org/10.1016/j.infsof.2014.05.018`.

[23] Hendrikx, M., Meijer, S., Van Der Velden, J., Iosup, A., 2013. Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl. 9. doi:`10.1145/2422956.2422957`.

[24] Karac, E.I., Turhan, B., Juristo, N., 2019. 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 .

[25] Kästner, C., Apel, S., Kuhlemann, M., 2008. Granularity in software product lines, in: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, ACM. pp. 311–320. doi:`10.1145/1368088.1368131`.

[26] Kasurinen, J., 2016. Games as software: Similarities and differences between the implementation projects, in: Proceedings of the 17th International Conference on Computer Systems and Technologies 2016, pp. 33–40.

[27] Kehrer, T., Thüm, T., Schultheiß, A., Bittner, P.M., 2021. Bridging the gap between clone-and-own and software product lines, IEEE Press. p. 21–25. doi:`10.1109/ICSE-NIER52604.2021.00013`.

[28] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., Emam, K.E., Rosenberg, J., 2002. Preliminary guidelines for empirical research in software engineering. IEEE Transactions on Software Engineering 28, 721–734.

[29] Krüger, J., Fenske, W., Thüm, T., Aporius, D., Saake, G., Leich, T., 2018. Apo-games: a case study for reverse engineering variability from cloned java variants, ACM. pp. 251–256. doi:`10.1145/3233027.3236403`.

[30] Krüger, J., Nell, L., Fenske, W., Saake, G., Leich, T., 2017. Finding lost features in cloned systems, Association for Computing Machinery, New York, NY, USA. p. 65–72. doi:`10.1145/3109729.3109736`.

[31] Lima, C., Assunção, W.K.G., Martinez, J., do Carmo Machado, I., von Flach G. Chavez, C., Mendonça, W.D.F., 2018a. Towards an automated product line architecture recovery: The apo-games case study, ACM. pp. 33–42. doi:`10.1145/3267183.3267187`.

[32] Lima, C., do Carmo Machado, I., de Almeida, E.S., von Flach G. Chavez, C., 2018b. Recovering the product line architecture of the apo-games, ACM. pp. 289–293. doi:`10.1145/3233027.3236398`.

[33] Lima, C., Chavez, C., de Almeida, E.S., 2017. Investigating the recovery of product line architectures: An approach proposal, Springer International Publishing, Cham. pp. 201–207.

[34] Liu, J., Snodgrass, S., Khalifa, A., Risi, S., Yannakakis, G.N., Togelius, J., 2020. Deep learning for procedural content generation. Neural Computing and Applications 33, 19–37. doi:`10.1007/s00521-020-05383-8`.

[35] Martinez, J., Tërnava, X., Ziadi, T., 2018. Software product line extraction from variability-rich systems: The robocode case study, Association for Computing Machinery, New York, NY, USA. p. 132–142. doi:`10.1145/3233027.3233038`.

[36] McShaffry, M., 2003. Game Coding Complete. Paraglyph Publishing.

[37] Moody, D.L., 2003. The method evaluation model: a theoretical model for validating information systems design methods. ECIS

2003 proceedings , 79.

[38] Moreira, R.A.F., Assunção, W.K., Martinez, J., Figueiredo, E., 2022. Open-source software product line extraction processes: the argouml-spl and phaser cases. Empirical Software Engineering 27.

[39] Panach, J.I., España, S., Dieste, Ó., Pastor, Ó., Juristo, N., 2015. In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. Information and Software Technology .

[40] Pascarella, L., Palomba, F., Penta, M.D., Bacchelli, A., 2018. How is video game development different from software development in open source?, ACM. pp. 392–402. doi:10.1145/3196398.3196418.

[41] Pohl, K., Metzger, A., 2018. Software Product Lines. Springer International Publishing, Cham. pp. 185–201. doi:10.1007/978-3-319-73897-0_11.

[42] Politowski, C., Petrillo, F., Montandon, J.E., Valente, M.T., Guéhéneuc, Y.G., 2021. Are game engines software frameworks? a three-perspective study. Journal of Systems and Software 171, 110846.

[43] Sarinho, V.T., Apolinário, A.L., Almeida, E.S., 2012. A feature-based environment for digital games, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 518–523.

[44] Selic, B., 2003. The pragmatics of model-driven development. IEEE Software 20, 19–25.

[45] Sierra, M., Pabón, M.C., Rincón, L., Newball, A.A.N., Linares, D., 2019. A comparative analysis of game engines to develop core assets for a software product line of mini-games, Springer. pp. 64–74. doi:10.1007/978-3-030-22888-0_5.

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

[47] Summerville, A.J., Snodgrass, S., Guzdial, M.J., Holmgård, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J., 2018. Procedural content generation via machine learning (pcgml). IEEE Transactions on Games 10, 257–270.

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

[49] Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C., 2011. Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games 3, 172–186. doi:10.1109/TCIAIG.2011.2148116.

[50] Trasobares, J.I., Domingo, Á., Arcega, L., Cetina, C., 2022. Evaluating the benefits of software product lines in game software engineering, ACM. pp. 120–130. doi:10.1145/3546932.3546998.

[51] Vaudour, F., Heinze, A., 2020. Software as a service: Lessons from the video game industry. Global Business and Organizational Excellence 39, 31–40.

[52] Vegas, S., Apa, C., Juristo, N., 2015. Crossover designs in software engineering experiments: Benefits and perils. IEEE Transactions on Software Engineering 42, 120–135.

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

[54] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in software engineering. Springer Science & Business Media.