Evaluating the Benefits of Software Product Lines in Game Software Engineering

Jose Ignacio Trasobares, África Domingo, Lorena Arcega, Carlos Cetina
jtrasobaresibor@acm.org,{adomingo,larcega,ccetina}@usj.es
Universidad San Jorge. Escuela de Arquitectura y Tecnología
Zaragoza, Spain

ABSTRACT

Video game development is one of the fastest-growing industries in the world. The use of software product lines (SPLs) has proven to be effective in developing different types of software at a lower cost, in less time, and with higher quality. There are recent research efforts that propose to apply SPLs in the domain of video games. Video games present characteristics that differentiate their development from the development of classic software; for example, game developers perceive more difficulties than other non-game developers when reusing code. In this paper, we evaluate if the adoption of an SPL in game software engineering (GSE) can generate the same benefits as in classic software engineering (CSE) considering the case study of Kromaia. As in other disciplines dealing with human behaviour, empirical research allows for building a reliable knowledge base in software engineering. We present an experiment comparing two development approaches, Clone and Own (CaO) and an SPL in terms of correctness, efficiency, and satisfaction when subjects develop elements of a commercial video game. The results indicate that the elements developed using the SPL are more correct than those developed with CaO but do not indicate significant improvement in efficiency or satisfaction. Our findings suggest that SPLs in GSE may play a different role than the one they have played for decades in CSE. Specifically, SPLs can be relevant to generating new video game content or to balancing video game difficulty.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; • **General and reference → Empirical studies**.

KEYWORDS

Empirical comparison, Software Product Line Engineering, Game Software Engineering

ACM Reference Format:
Jose Ignacio Trasobares, África Domingo, Lorena Arcega, Carlos Cetina. 2022. Evaluating the Benefits of Software Product Lines in Game Software Engineering. In *26th ACM International Systems and Software Product Line Conference - Volume A (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3546932.3546998

1 INTRODUCTION

Nowadays, the video game industry is one of the fastest-growing industries in the world. According to a 2019 report [31], the total number of active software developers is 18.9M. The same report indicates that the video game industry is responsible for 8.8M active developers. This means that almost half of the active developers are involved in the video game sector.

Nowadays, the majority of video games are developed using game engines. A game engine is a development environment that integrates two engines for graphics and physics and a set of tools to accelerate the development. The most popular are Unity [33] and Unreal Engine [15], but it is also possible for a studio to make its specific engine (e.g., CryEngine [9]).

Software models are a key artifact of game engines. Software models raise the abstraction level using terms that are much closer to the problem domain. This means that developers can focus on the game content itself, avoiding the implementation details of physics and graphics. Game engines allow video game developers to create content directly using code (e.g., C++) or software models. The code allows developers to have more control over the content. Although Unity and Unreal propose their own modeling language, a recent survey in Model-Driven Game Development [38] reveals that UML and Domain Specific Language (DSL) models are also adopted by development teams.

The use of software product lines (SPLs) has proven to be effective in developing different types of software at a lower cost, in less time, and with higher quality [29]. That is why there are recent research efforts that propose to apply SPLs in the domain of video games [19, 22, 30]. Another recent research [28] provided evidence that video game development is different from classical software development. For example, game developers perceive more difficulties than non-game developers when reusing code.

In this paper, we evaluate whether the adoption of an SPL in Game Software Engineering (GSE) can generate the same benefits as in classical software engineering (CSE) by analyzing the case study of a commercial video game: Kromaia. Kromaia has been previously analyzed by other authors since it uses a specific DSL to automatically generate the final bosses of the video game [4, 5, 12, 13]. We present an experiment in which we compare two development approaches, Clone and Own (CaO) and an SPL, both based on the Kromaia DSL, in terms of correctness, efficiency, and satisfiability. A total of 28 subjects (classified into two groups according to their experience) performed the tasks of the experiment, developing two final bosses of Kromaia. Although the bosses developed with SPL are more correct than those developed with CaO, our results do not indicate significant changes in efficiency or satisfaction.

Our results suggest that the efficiency that for decades has made SPLs attractive to CSE may not be the key to GSE. In light of our results, we may need to rethink the role of SPLs for GSE. Our work suggests new research directions for SPL in GSE. Specifically, our work reveals that SPLs in GSE can be relevant to generating new video game content (one of the hot topics of video game research) and to balancing difficulty (one of the seminal problems of video games).

The remainder of the paper is structured as follows. Section 2 presents the case study (Kromaia) and the SPL used in the experiment. 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 BACKGROUND

We use the video game Kromaia[1] in our experiment. Kromaia is a three-dimensional space game 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 the level, the player has to explore floating structures, avoid asteroids and find items. Along the way, the player will encounter basic enemies that will try 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 to complete the level.

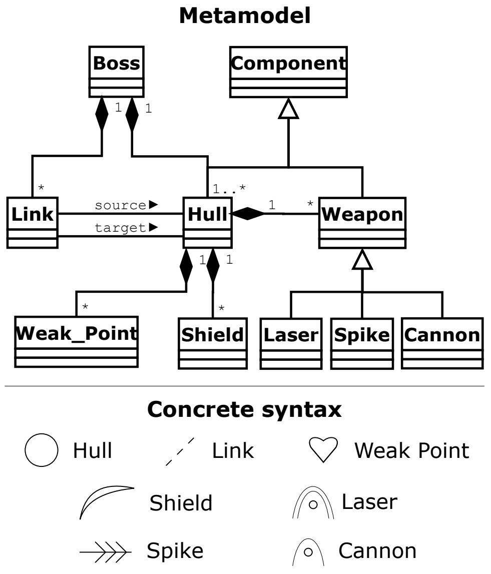

Figure 1: The Shooter Definition Modeling Language (SDML).

The final bosses are specified using the Shooter Definition Modeling Language (SDML) [5]. 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.

Figure 1 shows the metamodel and the concrete syntax of SDML. The metamodel shown in the upper part of the figure is a simplified version of the entire metamodel of SDML. This simplification omits concepts, relationships, and properties that are not as relevant as those presented in the figure. The metamodel contains more than 20 concepts, over 20 relationships, and more than 60 properties. However, this simplified version is complete enough to understand the elements and the relationships between them that compose the structure of a boss. The bottom part of Figure 1 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.

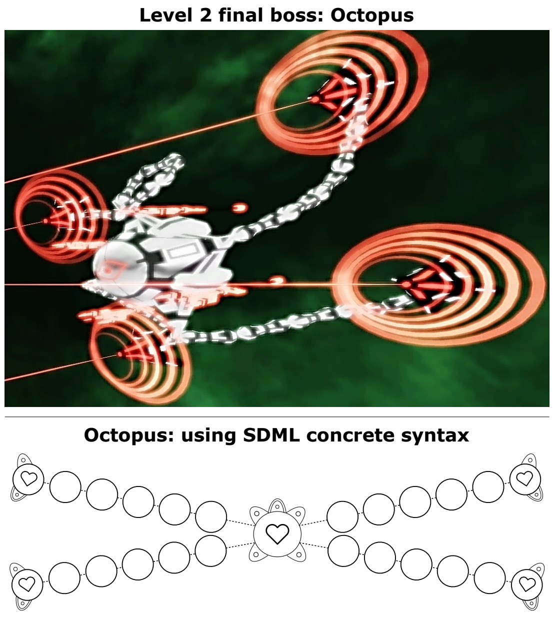
Level 2 final boss: Octopus
Octopus: using SDML concrete syntax

Figure 2: Example of a final boss of Kromaia.

An actual example of a final boss of Kromaia is presented in the upper part of Figure 2. The Octopus is the final boss that the player must defeat to complete level 2. The bottom part of the figure shows the model of the boss using the SDML concrete syntax. This boss is composed of one main hull (in the center of the model) and four "tentacles". The main hull has a weak point, a laser weapon, and four cannon weapons. Each tentacle consists of six hulls, where the end hull has a weak point, a laser weapon, and a cannon weapon.

Figure 3 shows a subset of the assets of the SPL for developing Kromaia bosses. In the left top corner, the figure shows one of the initial models using the concrete syntax of SDML that can be used to create new bosses. The right upper part of the figure shows some of the features that form the SPL. Each feature corresponds to a model fragment expressed using the concrete syntax of SDML. At the bottom of Figure 3, some feature models that represent the variability specification of a boss are shown. The variability

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

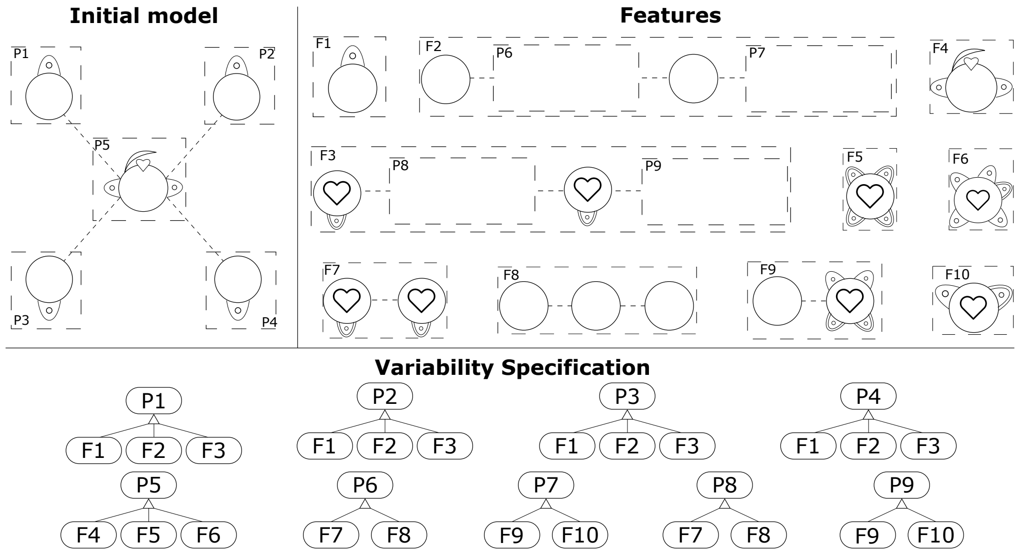

Figure 3: An example of the SPL for developing Kromaia bosses.

specification was defined by the domain experts of Kromaia. 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 place (*P1*) indicates that can be substituted with the model fragments *F1*, *F2*, or *F3*. Furthermore, some features can be variation points that have to be fulfilled with another feature. For example, feature *F2* has two variation points, *P6* and *P7*, which have to be substituted following their variability specification. *P6* has to be replaced by *F7* or *F8* and *P7* has to be replaced by *F9* or *F10*.

3 EXPERIMENT DESIGN

3.1 Objectives

According to the Wohlin's guidelines [37] for reporting software engineering experiments, we have organized our research objectives using the Goal Question Metric template for goal definition [3]. Our goal is to **analyze** different development approaches **for the purpose of** comparison, **with respect to** the correctness of the models constructed, efficiency, and user satisfaction, **from the point of view of** inexperienced and experienced developers, **in the context of** developing for a video game company.

3.2 Variables

In this study, the factor under investigation is **Development Approach**. There are two alternatives: to use Clone and Own (CaO) or the SPL based on the DSL of Kromaia, to create a boss of the video game.

Since the goal of this experiment is to evaluate the effects of the use of different approaches when developing a boss of a commercial video game, we selected *Correctness* and *Efficiency* as the objective dependent variables, which are related to performance. We measured *Correctness* using a correction template, which was applied to the models 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 time of each task. *Efficiency* is the ratio of *Correctness* to time spent (in minutes) to perform a task.

We also analyzed development approaches with respect to *Satisfaction* using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [25]. 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 language would require less effort. *Perceived Usefulness* (PU), the degree to which a person believes that using a particular language will increase performance, and *Intention to Use* (ITU), the degree to which a person intends to use a development approach. 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.

3.3 Design

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

These repeated measures design increases the sensitivity of the experiment [35]: 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 used CaO first and SPL afterwards, and G2 used SPL first and CaO afterwards) and different tasks, the design counterbalances some of the effects caused by using the alternatives of the factor in a specific order (i.e., learning effect, fatigue). To verify the experiment design, we conducted a pilot study with two subjects. This pilot study allowed us to estimate the time needed to solve the tasks and complete the questionnaires, detect typographical and semantic mistakes, and test the online environment used to develop the experiment. The subjects in the pilot study did not participate in the experiment.

3.4 Research questions and hypotheses

The research questions and null hypotheses are formulated as follows:

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

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

RQ3 - Is user satisfaction different when developers use different **Development Approach** for creating software for video games? To answer this question, we formulated three hypotheses based on the variables *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*, with their corresponding null hypotheses. These are: $H_{0,PEOU}$, the **Development Approach** does not have an effect on *Perceived Ease of Use*; $H_{0,PU}$, the **Development Approach** does not have an effect on *Perceived Usefulness*; $H_{0,ITU}$, the **Development Approach** does not have an effect on *Intention to Use*.

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

3.5 Participants

We selected the subjects using convenience sampling [37]. We invited 20 professionals working in software modeling or video game development to participate in the experiment. 13 of them decided to participate and completed the experiment. A total of 28 subjects with different knowledge about modeling and video game development performed the experiment. We also invited 20 third-year undergraduate students (inexperienced developers) taking a course in mobile game development from a technology program at a University. Of these subjects, 16 decided to participate and 15 completed the tasks and forms. The experiment was conducted by two instructors and one expert in the video game software domain. The expert provided information about the domain, the Kromaia DSL, and the SPL used in the experiment. This expert was not the same person who was responsible for designing the tasks. During the experiment, one of the instructors gave instructions and managed the focus groups. The other instructor clarified doubts about the experiment and took notes during the focus group.

3.6 Experimental objects

The tasks of our experiment were extracted from a real-world software development, Kromaia, which is a commercial video game released on PlayStation 4 and Steam. In Kromaia, the models are interpreted at run-time to create the C++ games' objects [5]. The tasks consisted of developing two final bosses of Kromaia from their behaviour in a gameplay video. The subjects used a graphical version of Shooter Definition Modeling Language, which is the DSL used in Kromaia. In one of the tasks of the experiment, the subjects used an SPL based on that DSL. A video game software engineer, involved in the development of Kromaia's DSL, designed the two tasks of similar difficulty and prepared the correction templates.

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

I Informed consent that subjects must review and accept voluntarily. It clearly explains what the experiment consists of and what will be the treatment of personal data.
II Demographic questionnaire to characterize the sample.
III Specific questionnaire to collect the subjects' responses during the experiment (their tasks, their times, and their answers to the satisfaction questionnaire). This part is different for each sequence. In one of the sequences, the subjects will perform the first task using the CaO approach and in the other sequence, they will perform the first task using the SPL approach. The second task will be performed with the approach not used in the first task.

The experimental objects used in this experiment (which includes the training material, the tasks, and the forms used for the questionnaires), as well as the results and the statistical analysis, are available at http://svit.usj.es/SPLvsCAO

3.7 Experimental procedure

The experiment was conducted on two different days. On the first day, the experiment was conducted online with professionals (experienced subjects). During the online session, all of the participants joined the same video conference via Microsoft Teams, and the chat session was used to clarify doubts or share information. On the second day, the experiment was conducted face-to-face with the group of students (inexperienced subjects). The experiment, scheduled for one hour and 45 minutes, was conducted following the experimental procedure described as follows:

(1) An instructor explained the parts of the session, and clarified to them that it was not a test of their abilities.
(2) The subjects attended a tutorial about the video game bosses to be developed and about the DSL and the SPL to be used in the experiment. The time devoted to this tutorial was 10 minutes. The information used was available to the subjects during the experiment. This information contained the metamodel, the concrete syntax of SDML (Figure 1), different examples of models of other video game bosses (such as the one in Figure 2), and an example of how to use the SPL.
(3) The subjects received clear instructions on where to find the links to access the forms for the experiment. They were also told about the structure of these forms and where they could find information about the DSL and the SPL. The subjects were randomly divided into two groups (G1 and G2); the subjects from G1 received the links to access one form and the subjects from G2 received a link to another form.
(4) The subjects accessed the online form, 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) The subjects completed a demographic questionnaire.
(6) The subjects performed the first task. The subjects from G1 had to use the CaO approach to develop a boss of the video game, and the subjects from G2 had to develop the same boss using the SPL. The subjects using the SPL approach had access to an initial model, a set of features, and the specification of the variability of the boss to be developed, as in the example of Figure 3. The subjects using the CaO approach did not have access to this information. After submitting their solution, the subjects completed a satisfaction questionnaire about the approach used for developing.
(7) The subjects performed the second task. The subjects from G1 developed another boss of the video game using the SPL approach, and the subjects from G2 developed the same boss using the CaO approach. Then, the subjects completed the satisfaction questionnaire.
(8) A focus group interview about the tasks (lasting 10 to 15 minutes) was conducted by one instructor while the other instructor took notes.
(9) Finally, the tasks were corrected, and a researcher analyzed the results.

3.8 Analysis procedure

We have chosen the Linear Mixed Model (LMM) [36] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows us to study the effects of factors that intervene in a crossover design (period, sequence, or subject) and the effects of other blocking variables (e.g., in our experiment, professional experience) [35]. 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 LDA 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, **Development Approach** (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 repeated measures design. The dependent variables (DV) for this test were *Correctness* and *Efficiency*, and the three other variables correspond to *Satisfaction*: *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to Use* (ITU).

In order to take into account the potential effects of factors that intervene in a crossover design in determining the main effect of **Development Approach**, we considered *Period* and *Sequence* to be fixed effects. In order to explore the potential effects of the subject's experience to determine the variability in the dependent variables, we also considered in the statistical model the fixed factors **Experience** and the sequence **Development Approach** and **Experience**.

We tested different statistical models in order to find out which factors, in addition to **Development Approach**, 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, **Development Approach**. We also tested other statistical models (e.g., model 1, 2, 3, or 4) that included other fixed factors (experience, period, or sequence) that could have effects on the dependent variables.

$$\begin{array}{ll}(Model\ 0) & DV \sim DA + (1|Subj.)\\ (Model\ 1) & DV \sim DA + Experience + (1|Subj.)\\ (Model\ 2) & DV \sim DA + Experience + DA * Experience + (1|Subj.)\\ (Model\ 3) & DV \sim DA + Experience + Period + (1|Subj.)\\ (Model\ 4) & DV \sim DA + Sequence + Period + (1|Subj.)\end{array} \quad (1)$$

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 [13, 17]. The assumption for applying LMM is the normality of the residuals of the dependent variables. In the analysis, we only consider statistical models verifying this assumption. To verify this normality, we used Kolmogorov-Smirnov tests as well as visual inspections of the histogram and normal Q-Q plots. To describe the changes in each dependent variable, we selected the statistical model that satisfied the normality of residuals and also obtained the smallest AIC or BIC value.

To quantify the differences in the dependent variables due to significant fixed factors, we calculated the Cohen d value [8] between the alternatives of these factors. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. We selected histograms and box plots to graphically describe the data and the results.

4 RESULTS

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 1 shows the number of subjects in the experiment, divided by experience, 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). The mean and standard deviation of their answers 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	28	25.8±7.4	3.4±2.5	0.8±1	3±1.3	2.6±1.6
Experienced	13	30.7±8.4	4.5±2.6	1.2±1.2	3.8±1.3	3.5±1.4
Inexperienced	15	21.5±1.3	2.4±2	0.5±0.6	2.3±0.7	1.7±1.2

4.1 Changes in the dependent variables

There were differences in the means and standard deviations of all of the dependent variables depending on which **Development Approach** was used to create a boss of a video game. However, the differences in *Effectiveness* and *Satisfaction* are very small. Table 2 shows the values for the mean and standard deviation 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

	Development Approach	Experience		Period		Sequence	
		Experienced	Inexperienced	Task 1	Task 2	G1 Cao-SPL	G2 SPL-CaO
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	4.5±2.29	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	3.27±1.34	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	3.93±0.72	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	3.89±0.69	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	3.66±1.11	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	3.5±1.28	2.77±1.33	2.77±1.33	3.85±0.9

the dependent variables *Correction*, *Efficiency*, *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to Use* (ITU) for each one of the **Development Approaches** compared: CaO and SPL, and for each one of the alternatives of the fixed factors considered in the statistical analysis: **Experience**, with two alternatives (Experienced and Inexperienced subjects); for **Period** with two alternatives, each one associated with one of the tasks (Task 1, and Task 2), and **Sequence**, whose two alternatives reflect the order in which subjects have used the development approaches, (G1: CaO-SPL, G2: SPL-CaO).

To quantify the differences in the dependent variables due to each factor, we analyzed the Cohen *d* values. Table 3 shows the Cohen *d* values of the dependent variables for all of the fixed factors considered in the statistical analysis. Values indicating medium or high variation due to the factor are highlighted in bold, and those corresponding to significant differences according to the hypothesis tests are shaded in grey. Positive values indicate differences in favor of the first alternative of the factors and negative values indicate differences in favor of the second alternative of the factor. According to the Cohen *d* values of the dependent variables for **Development Approach** (first column of Table 3), we can affirm that the effect size of the factor **Development Approach** in favor of SPL for *Correctness* was medium, with a Cohen *d* value of -0.516, and small for *Perceived Ease of Use*, with a Cohen *d* value of -0.251. However, the effect of **Development Approach** on *Efficiency*, *Perceived Usefulness*, and *Intention to Use* is negligible with Cohen *d* values of less than 2 in favor of CaO. Table 3 also shows the Cohen *d* values of the dependent variables for the rest of the fixed factors considered in the statistical analysis. These values indicate that the factor **Experience** has large effects on *Correctness*, *Perceived Usefulness*, and *Intention to Use* in favor of experienced subjects. The factor **Period** has medium effects on *Efficiency*, *Perceived Ease of Use*, and *Intention to Use*. The subjects were more efficient in the second task, and *Perceived Ease of Use* and *Intention to Use* were better valued in the first task. The factor **Sequence** has a small effect on all dependent variables, except for Efficiency, for which the effect of sequence is medium. Subjects who started using SPL in the first task performed better on both tasks than subjects who started with CaO.

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

	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

The Cohen *d* values are related to the percentage of non-overlap between the distributions of the dependent variables for each fixed factor. Higher values correspond with greater percentages of non-overlap and larger differences. The histograms in Figure 4 illustrate the differences in *Correctness* depending on the factor **Development Approach**. In the histograms, the non-overlapping parts have a single pattern (either dots or shaded), while the overlapping parts have both patterns (dots and shaded). The non-overlapping parts for *Correctness* are around 20%, which corresponds to a medium effect size. The box plots of Figure 4 corresponding to all subjects show the differences in *Correctness* due to **Development Approach**. The median of *Correctness* is 20% higher for SPL than for CaO, but also the dispersion increases when subjects use the SPL approach. The box plots of Figure 4 corresponding to experienced and inexperienced subjects, show the differences in *Correctness* due to the **Experience** and **Development Approach**. Both experienced and inexperienced subjects develop more correct models when using SPL instead of CaO. However, regardless of the development approach used, models developed by experienced subjects were more correct than those developed by inexperienced subjects.

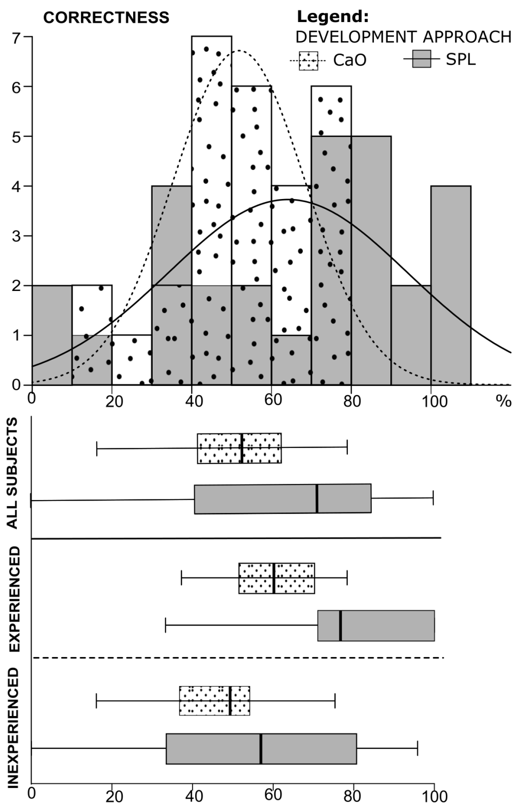

Figure 4: Histograms with normal distributions and box plots for Correctness

Figure 5 illustrates the differences in *Efficiency* depending on the factor **Development Approach**. The non-overlapping parts for *Efficiency* are less than 3%, which corresponds to a negligible effect between the use of CaO or SPL. The box plots of Figure 5 corresponding to all subjects show negligible differences in *Efficiency* due to **Development Approach**. The box plots of Figure 5 corresponding to the alternatives of **Period** (Task 1 and Task 2) show the differences in *Efficiency* due to **Development Approach** and **Period**. The subjects were more efficient in Task 2 than in Task 1, mainly when using the CaO development approach. For the dependent variables related to *Satisfaction* (*PEOU*, *PU*, and *ITU*) the representation of the differences due to **Development approach** would be more similar to that shown for *Efficiency* than that shown for *Correction*. Differences due to **Experience** in *PU* and *ITU*, in favor of experienced subjects, would be represented with boxplots like those shown for *Correction*. Boxplots equivalent to the latter would also illustrate the differences in favor of Task 1 due to **Period** in *PEOU* and *ITU*.

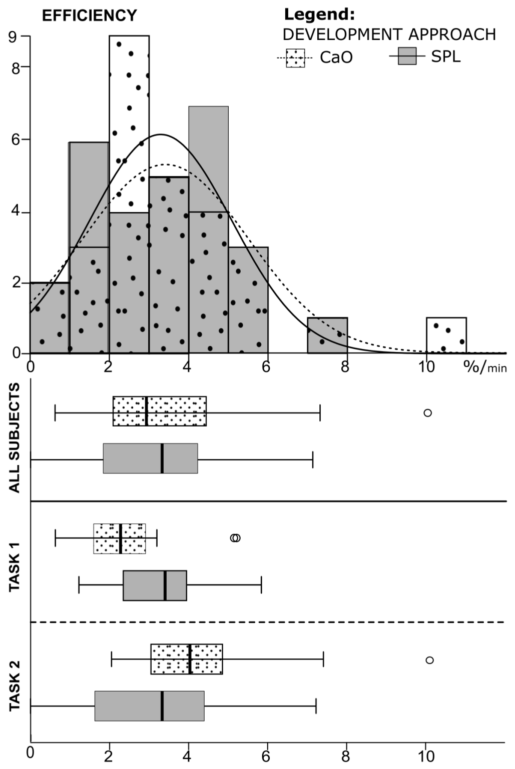

Figure 5: Histograms with normal distributions and box plots for Efficiency

The Linear Mixed Model used to study the statistical significance of the changes in the dependent variables *Correction* and *Efficiency*, was the one given by formula 2:

$$\begin{aligned} DV \sim DA + Experience + Period + Sequence + \\ DA * Experience + Experience * Period + \\ Experience * Sequence + (1|Subj.) \end{aligned} \quad (2)$$

To study the statistical significance of the changes in *Satisfaction* (*PEOU*, *PU*, and *ITU*) the statistical model chosen was the one given by formula 3.

$$\begin{aligned} DV \sim DA + Experience + Period + DA * Experience + \\ DA * Period + Experience * Period + (1|Subj.) \end{aligned} \quad (3)$$

Models such as those in formula 1, and others LMM tested, did not verify the normality of the residuals or obtained higher values for the AIC and BIC fit statistics than those obtained by the selected models. The factors and combinations of factors that are part of formulas 2 and 3 explain the changes in the dependent variables; however, according to the hypothesis test results, not all the changes in the dependent variables due to these factors are significant. Table 4 shows the results of the Type III fixed effects test for each of the dependent variables and for each fixed factor of the statistical model used in each case. Values indicating significant differences are shaded in grey.

Table 4: Results of Type III test of fixed effects for each variable and factor. NA=Not Applicable

	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

For *Correctness*, the factor **Development Approach** obtained a p-value of less than 0.05. Therefore, our first null hypothesis is rejected. Thus, the answer to **RQ1** is affirmative. The **Development Approach** used for creating a boss in a video game has a significant impact on *Correctness*. The results of the hypothesis tests confirm the statistical significance of the differences observed between the descriptive statistics and the graphical representations. The models made with SPL are more correct than those made with CaO. In Addition, the factor **Experience**, and the combination of **Experience** and **Sequence** were considered to be statistically significant to explain the changes in *Correctness*. The effect of **Experience** in *Correctness* is large in favor of experienced subjects. Experienced subjects obtain more correct models than inexperienced subjects, both when using CaO and when using SPL. The descriptive statistics for the alternatives for the combination of the factors **Experience** and **Sequence**: Experienced-G1 ($\mu = 71.87; \sigma = 20.52$), Experienced-G2 ($64.17\mu; \sigma = 19.22$), Inexperienced-G2 ($\mu = 49.064; \sigma = 25.50$), and Inexperienced-G1 ($\mu = 40.44; \sigma = 26.04$) indicate large differences between experienced and inexperienced subjects whose first exercise was developed using CaO (Group 1). Experienced subjects in Group 1 obtain the best results in correctness considering both tasks (CaO and SPL), while inexperienced subjects in this group perform the least correct models, also in both tasks.

For *Efficiency*, the factor **Development Approach** obtained a p-value greater than 0.05. Therefore, we can not reject the second null hypothesis and the answer to the research question **RQ2** is negative. The **Development Approach** used does not have a significant impact on *Efficiency*. Subjects spend more time developing when using SPL ($\mu = 20.6$ minutes; $\sigma = 5.72$) instead of CaO ($\mu = 18.43 minutes; \sigma = 7.33$), which counteracts the positive effect of the use of SPL on *Correctness*. On the other hand, **Period** has a significant impact on *Efficiency*. Subjects are more efficient in the second task, which could indicate a learning effect.

For the dependent variables related to *Satisfaction*, the factor **Development Approach** obtained p-values greater than 0.05. Therefore, we can not reject the third null hypothesis and the answer to research question **RQ3** is negative. **Development Approach** does not have a significant impact on the satisfaction of subjects when developing a video game boss from a video game. However, there are significant changes in *Perceived Ease of Use* due to **Period**. Subjects perceive the development approach used in the first task, regardless of which one (CaO or SPL), as easier to use than the development approach used in the second task. Also, *Intention to Use* achieves higher values in the first task than in the second. In addition, the factor **Experience** has significant and large effects on satisfaction: experienced subjects score higher than inexperienced subjects on *Perceived Usefulness* and *Intention to Use* for either of the two approaches used in the experiment.

5 DISCUSSION

As recorded over the last two decades by the Software Engineering Institute of the Carnegie Mellon University[2], SPLs can multiply productivity by a factor of 10, reduce costs up to 60% and 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 the SPLs for Classic Software Engineering (CSE).

However, our results in the video game domain do not show significant differences in efficiency when using the SPL. Existing differences between CSE and Game Software Engineering (GSE) (e.g., working on different kinds of artifacts or how they perceive the development process of their projects) may mean that the efficiency is not the key selling point of SPLs for GSE. To better understand the experiment results, we carried out a focus group with the subjects. The focus group had open questions, such as: *What development approach did you like the most and for what would you use each of them?, Has the mental process or steps you followed to create the boss been the same with both approaches?.*

The focus group revealed that, regardless of whether the subjects used CaO or SPL, where they spent more time was evaluating in execution if what they were understanding was the boss they wanted to develop. Subjects stated that they needed to constantly check the video game in real-time to understand what they had to develop. The efficiency benefits of the SPL were not significant because the greatest effort belonged to the part of constantly checking the video game in real-time and this point is common to both CaO and SPL. Most of the subjects acknowledge that the SPL did not accelerate video game development.

Despite the above, most of the subjects observed benefits in SPL for video games. Specifically, most of the subjects mentioned that the SPL was very relevant to create 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, it is common that, once the video game is launched, it is kept alive through updates that are known as Downloadable Content (also called DLCs).

The subjects reported that thinking in terms of SPL features helped them to think of new content for the video game. The subjects valued positively that by combining the features following

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

the rules of the variability specification they obtained new content. This new content was indeed different but similar to what already existed in the video game, however, the subjects affirmed that this type of 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 to avoid repetition; it could also be used to fill in secondary parts of the game that were more sparsely populated; or even as a source of inspiration to create completely new content.

In the video game research community, content generation for video games is a hot topic. Many surveys [16, 23, 32, 34] talk about how to automate (completely or partially) the content generation. Specifically, the automation of content generation is known as Procedural Content Generation (PCG) [2] 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 games content is 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 in SPL only 10 decisions were needed, while in CaO he had to make between 50 and 100 decisions. 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.

Finally, a group of subjects reported that the SPL may be relevant to help 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.

6 THREATS TO VALIDITY

To describe the threats to validity of our work, we use the classification of Wohlin et al. [37]:

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 is 95%. To minimize the *fishing and the error rate* threat, the statistical analysis has been done by a researcher who did not participate in the task design or in the correction process. The *Reliability of measures* threat was mitigated because the 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 the two sessions. Also, the tasks were designed with similar difficulty.

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 *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience. In addition, the subjects 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. The *interactions with selection* threat also affected the experiment because of the voluntary nature of participation. To avoid student demotivation, we selected students from a course whose contents fit the design of the experiment.

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 [27]. 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 by using a widely applied model (TAM) [10]. 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 participation nor results would affect their grades in the subject. *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 instructors with similar difficulty for the two tasks. Finally, the *mono-operation bias* threat occurs when the treatments depend on a single operationalization. The experiment was affected by this threat since we worked with a specific DSL.

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 [18], as is the case in this experiment. In addition, the experience factor has been taken into account in the data analysis. The *domain* threat occurs because the experiment has been conducted in a specific domain, video game, and for a very specific type of task, i.e., to develop a video game boss from Kromaia. We think that the generalizability of findings should be undertaken with caution. Other experiments in different games should be performed to validate our findings.

7 RELATED WORK

We summarize related works taking into account the area of the Software Product Lines applied to the video game domain.

Lima et al. [20, 22] present two works about Product Line Architecture (PLA) recovery. In [22], they applied their previous proposed approach and guidelines [21] to create the PLA of the Apo-Games project [19]. They used a set of cloned Java and Android projects as input for recovering architectural information. They created a set of UML diagrams and Design Structure Matrices (DSMs) that described the PLA core elements and the variability at the architectural level. Similarly, in [20], they developed an automatic approach for the identification of the minimum subset of cross-product architectural information for an effective PLA recovery. They proposed the improvement of the recovered PLA by combining threshold analysis with the elimination of outliers and metrics analysis.

Boaventura and Sarinho [6] present Minimal Engine for Digital Games (MEnDiGa). MEnDiGa is a collection of game assets to create small video games. They develop minimal features, which are game logic features, that are related in a hierarchy. In addition, MEnDiGa has modules to represent, interpret, and adapt game features for functionality in multiple game platforms. To evaluate their work, they develop a clone of Doodle Jump and then compare the original with the clone. The results are that the core of game elements with MEnDiGa can be independent, reusable, and large-scale way.

Moreira et al. [26] provide and analyze empirical data of the extraction processes of two open-source case studies, namely ArgoUML and Phaser. Phaser [3] is an open-source 2D game framework for Web browsers. Their results show that there is a great difference between the re-engineering process of ArgoUML-SPL and Phaser. However, even though only a few developers were in charge of the re-engineering process, common problems were found in both cases, related to lack of tools that led to incomplete and inconsistent feature extractions, complexity on managing feature dependencies when using compositional approach, and issues of not having a variability model to deal with feature constraints.

Similarly, Martinez et al. [24] report an experience on 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, participants gained hands-on practice in implementing systematic reuse so they can respond to more advanced software engineering challenges in implementing families of systems, the extractive process, the process followed by the participants to implement a feature (select, analyse, implement and/or adapt, test and debug), and the analysis of the time and effort, around half of their time was spent in the feature implementation part.

Debbiche et al. [11] analyze the Apo-Games to identify reusable code or artifacts. They analyze 5 Java games and migrated 3 of these into a composition-based software product line implemented with *FeatureHouse* [1]. In addition, they present their lessons learned: many changes are required to enable composition which makes extracting a composition-based software product line challenging and time-consuming, they recommend making sure that the software product line is always testable to ensure the correct transformation to code, and incrementally adopting new features facilitates the extraction, as various artifacts need updates and must be tested.

Sierra et al. [30] present a comparison between two video game engines: *unity* and *P5*, in order to select the most appropriate option to implement the SPL core assets. Their comparison considers the performance, learning curve, and license terms of both engines as additional criteria to decide which engine to use. They use an SPL of video games that support oral and written rehabilitation therapies in children with hearing impairments. Through their comparison, they verify that using P5 streamlines the development and the generation times. They proved that the SPL paradigm and the reuse concept can be applicable, not only for classic software but also in the development of serious games.

Castro and Werner [7] present a prototype of a game that was developed by applying a software product line to generate mods systematically. A mod is a new version of a game that has been created by applying modifications or adaptations to an existing game. The prototype created makes use of the idea of dynamic product lines applied to mods. 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. In future work, they intend to evaluate the game with experts to validate the idea of creating mods using software product lines.

The above works apply software product line approaches in the video game domain. However, these works do not evaluate whether the benefits that produce the use of software product lines in classic software engineering apply in game software engineering. In this work, we evaluate the SPL with a commercial video game and with subjects linked to the development of video games. To the best of our knowledge, this is the first work that empirically evaluates the application of a software product line in game software engineering.

After analyzing these works, we realized that there is a lack of empirical studies comparing SPLs and CaO. We were able to find a paper that performed this comparison [14]. They compare the performances of software engineers in the software products development process using SPL and CaO in an industrial context. Their results achieve better values for effectiveness, efficiency, and satisfaction with the SPL approach. In contrast to our work, they are focused on Classic Software Engineering (CSE) and do not address Game Software Engineering (GSE).

8 CONCLUSION

In this paper, we present an experiment that evaluates whether the adoption of an SPL in GSE can generate the same benefits as in CSE [14]. In our experiment, we compare two development approaches, CaO and SPL. A total of 28 subjects performed the experiment tasks developing two final bosses of the case study, Kromaia.

We evaluate the effects of the usage of the two development approaches in terms of correctness, efficiency, and satisfaction. The results show that the bosses developed with the SPL are more correct than the bosses developed with CaO. However, there are not-significant changes in efficiency or satisfaction. It turns out that SPLs in game software engineering may play a different role than they have for decades in classic software engineering.

Our results reveal that SPLs in GSE can be relevant to generating new video game content (a hot topic in the video game research community) and to balancing the video game difficulty (a seminal problem of video games). Therefore, we hope that this work encourages further research and new research directions for SPLs in GSE.

ACKNOWLEDGMENTS

Iniciando Lineas de Producto Software Mediante Busquedas Interactivas Dirigidas por Modelos. Reference: PID2021-128695OB-I00.

[3]Phaser: http://phaser.io/

REFERENCES

[1] Sven Apel, Christian Kästner, and Christian Lengauer. 2013. Language-Independent and Automated Software Composition: The FeatureHouse Experience. *IEEE Transactions on Software Engineering* 39, 1 (2013), 63–79. https://doi.org/10.1109/TSE.2011.120

[2] Nicolas A. Barriga. 2019. A Short Introduction to Procedural Content Generation Algorithms for Videogames. *International Journal on Artificial Intelligence Tools* 28, 02 (2019), 1930001. https://doi.org/10.1142/S0218213019300011

[3] Victor R. Basili and H. Dieter Rombach. 1988. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering* (1988).

[4] Daniel Blasco, Carlos Cetina, and Oscar Pastor. 2020. A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study. *Inf. Softw. Technol.* 119 (2020). https://doi.org/10.1016/j.infsof.2019.106235

[5] Daniel Blasco, Jaime Font, Mar Zamorano, and Carlos Cetina. 2021. An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering. *Journal of Systems and Software* 171 (2021), 110804.

[6] Filipe M. B. Boaventura and Victor Travassos Sarinho. 2017. MEnDiGa: A Minimal Engine for Digital Games. *Int. J. Comput. Games Technol.* 2017 (2017), 9626710:1–9626710:13. https://doi.org/10.1155/2017/9626710

[7] Diego Castro and Cláudia Werner. 2021. Rebuilding games at runtime. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021 - Workshops, Melbourne, Australia, November 15-19, 2021*. IEEE, 73–77. https://doi.org/10.1109/ASEW52652.2021.00025

[8] Jacob Cohen. 1988. Statistical power for the social sciences. *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988).

[9] Crytek. 2002. CRYENGINE | The complete solution for next generation game development by Crytek. https://www.cryengine.com. [Online; accessed 21-November-2021].

[10] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Q.* 13, 3 (Sept. 1989), 319–340.

[11] Jamel Debbiche, Oskar Lignell, Jacob Krüger, and Thorsten Berger. 2019. Migrating Java-based Apo-Games into a composition-based software product line. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, 18:1–18:5. https://doi.org/10.1145/3336294.3342361

[12] África Domingo, Jorge Echeverría, Oscar Pastor, and Carlos Cetina. 2020. Evaluating the Benefits of Model-Driven Development - Empirical Evaluation Paper. In *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12127)*, Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant (Eds.). Springer, 353–367. https://doi.org/10.1007/978-3-030-49435-3_22

[13] África Domingo, Jorge Echeverría, Óscar Pastor, and Carlos Cetina. 2021. Comparing UML-based and DSL-based Modeling from Subjective and Objective Perspectives. In *International Conference on Advanced Information Systems Engineering*. Springer, 483–498.

[14] Jorge Echeverría, Francisca Pérez, José Ignacio Panach, and Carlos Cetina. 2021. An empirical study of performance using Clone & Own and Software Product Lines in an industrial context. *Inf. Softw. Technol.* 130 (2021), 106444. https://doi.org/10.1016/j.infsof.2020.106444

[15] Epic Games. 1998. Unreal Engine: The most powerful real-time 3D creation tool. https://www.unrealengine.com. [Online; accessed 21-November-2021].

[16] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.* 9, 1, Article 1 (feb 2013), 22 pages. https://doi.org/10.1145/2422956.2422957

[17] Evrim Itir Karac, Burak Turhan, and Natalia Juristo. 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* (2019).

[18] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28, 8 (Aug 2002), 721–734.

[19] Jacob Krüger, Wolfram Fenske, Thomas Thüm, Dirk Aporius, Gunter Saake, and Thomas Leich. 2018. Apo-games: a case study for reverse engineering variability from cloned Java variants. In *Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, Thorsten Berger, Paulo Borba, Goetz Botterweck, Tomi Männistö, David Benavides, Sarah Nadi, Timo Kehrer, Rick Rabiser, Christoph Elsner, and Mukelabai Mukelabai (Eds.). ACM, 251–256. https://doi.org/10.1145/3233027.3236403

[20] Crescencio Lima, Wesley K. G. Assunção, Jabier Martinez, Ivan do Carmo Machado, Christina von Flach G. Chavez, and Willian Douglas Ferrari Mendonça. 2018. Towards an Automated Product Line Architecture Recovery: The Apo-Games Case Study. In *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS 2018, Sao Carlos, Brazil, September 17-21, 2018*. ACM, 33–42. https://doi.org/10.1145/3267183.3267187

[21] Crescencio Lima, Christina Chavez, and Eduardo Santana de Almeida. 2017. Investigating the Recovery of Product Line Architectures: An Approach Proposal. In *Mastering Scale and Complexity in Software Reuse*, Goetz Botterweck and Claudia Werner (Eds.). Springer International Publishing, Cham, 201–207.

[22] Crescencio Lima, Ivan do Carmo Machado, Eduardo Santana de Almeida, and Christina von Flach G. Chavez. 2018. Recovering the product line architecture of the apo-games. In *Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*, Thorsten Berger, Paulo Borba, Goetz Botterweck, Tomi Männistö, David Benavides, Sarah Nadi, Timo Kehrer, Rick Rabiser, Christoph Elsner, and Mukelabai Mukelabai (Eds.). ACM, 289–293. https://doi.org/10.1145/3233027.3236398

[23] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. 2020. Deep learning for procedural content generation. *Neural Computing and Applications* 33, 1 (oct 2020), 19–37. https://doi.org/10.1007/s00521-020-05383-8

[24] Jabier Martinez, Xhevahire Tërnava, and Tewfik Ziadi. 2018. Software Product Line Extraction from Variability-Rich Systems: The Robocode Case Study. In *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1* (Gothenburg, Sweden) *(SPLC '18)*. Association for Computing Machinery, New York, NY, USA, 132–142. https://doi.org/10.1145/3233027.3233038

[25] Daniel L Moody. 2003. The method evaluation model: a theoretical model for validating information systems design methods. *ECIS 2003 proceedings* (2003), 79.

[26] Rodrigo André Ferreira Moreira, Wesley KG Assunção, Jabier Martinez, and Eduardo Figueiredo. 2022. Open-source software product line extraction processes: the ArgoUML-SPL and Phaser cases. *Empirical Software Engineering* 27, 4 (2022).

[27] Jose Ignacio Panach, Sergio España, Óscar Dieste, Óscar Pastor, and Natalia Juristo. 2015. In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. *Information and Software Technology* (2015).

[28] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. 2018. How is video game development different from software development in open source?. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM, 392–402. https://doi.org/10.1145/3196398.3196418

[29] Klaus Pohl and Andreas Metzger. 2018. *Software Product Lines*. Springer International Publishing, Cham, 185–201. https://doi.org/10.1007/978-3-319-73897-0_11

[30] Martín Sierra, María Constanza Pabón, Luisa Rincón, Andrés Adolfo Navarro Newball, and Diego Linares. 2019. A Comparative Analysis of Game Engines to Develop Core Assets for a Software Product Line of Mini-Games. In *Reuse in the Big Data Era - 18th International Conference on Software and Systems Reuse, ICSR 2019, Cincinnati, OH, USA, June 26-28, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11602)*, Xin Peng, Apostolos Ampatzoglou, and Tanmay Bhowmik (Eds.). Springer, 64–74. https://doi.org/10.1007/978-3-030-22888-0_5

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

[32] Adam James Summerville, Sam Snodgrass, Matthew J. Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10 (2018), 257–270.

[33] Unity Technologies. 2005. Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. https://unity.com. [Online; accessed 21-November-2021].

[34] Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

[35] Sira Vegas, Cecilia Apa, and Natalia Juristo. 2015. Crossover designs in software engineering experiments: Benefits and perils. *IEEE Transactions on Software Engineering* 42, 2 (2015), 120–135.

[36] Brady T West, Kathleen B Welch, and Andrzej T Galecki. 2014. *Linear mixed models: a practical guide using statistical software*. Chapman and Hall/CRC.

[37] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[38] Meng Zhu and Alf Inge Wang. 2020. Model-driven Game Development: A Literature Review. *ACM Comput. Surv.* 52, 6 (2020), 123:1–123:32. https://doi.org/10.1145/3365000