A Multiple Case Study on Reuse in Game Software Engineering

Jose Ignacio Trasobares[a,b,*], África Domingo[a], Rodrigo Casamayor[a], Daniel Blasco[a], Carlos Cetina[b]

[a]SVIT Research Group, Universidad San Jorge, Zaragoza, Spain
[b]Universitat Politècnica de València, Valencia, Spain

Abstract

Context: Game Software Engineering (GSE) is a specialized field at the intersection of software engineering and video game development. Reuse in GSE is particularly complex due to the iterative nature of game development and technical needs that arise in creating interactive digital experiences.

Objective: This paper presents the first multi-case study on reuse in GSE, focusing on how reusable components are developed and maintained in game projects. The study aims to investigate reuse practices by analyzing multiple sources, including access to game projects, interviews with developers, focus groups, studio visits, and code analysis.

Method: The study integrates various evidence sources to gain a comprehensive view of reuse in GSE. Data were gathered from interviews and focus groups, supplemented by direct observations during visits. Additionally, a recent proposal on software phylogenetics was applied to analyze source code, providing insights into reuse in game projects.

Results: Our findings highlight the significance of prefabs in promoting reuse, especially in managing complex game objects. Prefabs emerged as a widely used element, confirmed by developer feedback and repository analysis. Software phylogenetics also revealed certain drawbacks.

Conclusion: While prefabs play a relevant role enhance reusability, they can introduce redundancy, bugs, and unused components (dead prefabs). Understanding these limitations could inspire future research addressing such issues. Prefab-related practices in GSE could benefit other software engineering areas, encouraging broader reuse strategies.

Keywords: Game Software Engineering, Software Reuse, Multiple Case Study

1. Introduction

Video games are not only a work of art but also a software development. In fact, a report on the developer population reveals that 8.8 million out of the 18.9 million active software developers have worked on video games [1]. The relevance of software in video games has given rise to what is known as Game Software Engineering (GSE) [2, 3, 4]. GSE is a subfield that intersects Software Engineering and video games. A recent SLR on GSE reveals that reusability is one of the most relevant topics in GSE [3]. In fact, reusability is a cornerstone of software engineering, enabling developers to leverage existing components to reduce development time and improve maintainability [5].

To better understand the importance of reusability, it is essential to define the concept. The most widely accepted definition of software reuse [6] is: "Software reuse is the process of creating software systems from existing software rather than building software systems from scratch." In this context, we have analyzed two main types of reuse. On the one hand, reuse across different projects, where elements from one project are repurposed for others. On the other hand, reuse within the same project, where components developed at different stages of the same project are leveraged. Understanding these forms of reuse is crucial for evaluating how they manifest in the context of GSE.

Despite the importance of reuse in GSE, we have not found a clear answer to the question of how it is reused in the real context of video games in previous works. Previous works rely on either video game analysis or interviews, with none of them combining multiple sources of evidence. Furthermore, after analyzing the existing literature, we observe a lack of extensive research on reuse in GSE that analyzes the problem in the real context of video game development. To address this gap, our research presents the first multiple-case study that integrates multiple sources of evidence to provide a thorough examination of reuse in GSE.

Case studies are an empirical strategy that allows studying a contemporary phenomenon, using different methods of data collection, within a real-life context in order to answer explanatory research questions. However, not all of the works labeled as case studies use a real context, different methods of data collections, or study contemporary phenomenona [7, 8].

The work presented in this paper has a multiple case study design, according to the methodological frameworks proposed in the guidelines of Yin et al. [9] and Runeson et al. [10]. Our multiple-case study builds upon these frameworks, as they are widely accepted by the research community. These guide-

*Corresponding author.
Email addresses: jtrasobares@usj.es (Jose Ignacio Trasobares), adomingo@usj.es (África Domingo), rcasamayor@usj.es (Rodrigo Casamayor), dblasco@usj.es (Daniel Blasco), cetina@upv.es (Carlos Cetina)

lines describe the aspects that differentiate case studies from other empirical methods and discuss the different elements that should be taken into account in their design, execution and analysis through different examples. We designed our study to encompass multiple sources of evidence, including direct access to source projects, interviews with game developers, and an analysis of data using a phylogenetic perspective [5, 11]. Software Phylogenetics is a recent proposal [5, 11] that enables the visualization of the evolutionary relationships among software components, offering insights into their development and evolution within software projects.

Furthermore, recognizing that expectations and experiences in reuse could vary greatly, we found it important to study not just one unit of analysis, but multiple units, in order to achieve data triangulation. Triangulation increases precision and strengthens the validity of empirical research, especially when statements based on statistical significance are not possible. Triangulation means taking multiple perspectives towards the studied object thereby providing a broader picture [10]. We applied three types of triangulation: data source triangulation, by using more than one data source and collecting data on different occasions; observer triangulation, by using more than one observer in the study; and methodological triangulation, by combining different types of data collection methods: questionnaires, interviews, archival analysis such as questionnaires, interviews, archival analysis, focus groups and direct observations.

Our results confirm the claims of previous works regarding (1) the widespread use of game engines, (2) the motivation for reuse, and (3) the characteristics of development teams for reuse [12, 13, 14]. Our work also reveals the centrality of prefabs[1] in reuse. Prefabs are pre-configured GameObject templates that can be reused and customized, simplifying the creation and management of multiple instances of complex objects. Surprisingly prefabs had not been addressed, or even mentioned (either in the alternative forms of Blueprint class or scene) in previous studies on reuse in GSE [3].

Our analysis of source code repositories corroborates the prevalence of prefabs in game development. We observed a widespread use of prefabs in the projects, highlighting their key role in facilitating reuse. While prefabs are well-known by developers, they have not been extensively studied or discussed in the scientific literature as fundamental units of reuse. In addition, software phylogenetics offers new insights into the evolutionary patterns of prefabs, showing how these reusable units are organized within a game project. After discovering that reuse between different projects is neither mainstream nor fundamental in the GSE context, our focus will shift to reuse within a single game, leveraging phylogenetics as a tool to visualize and analyze the relationships and reuse patterns of its components. Our work also reveals a lack of understanding of the reuse relationships among prefabs by developers. Developers recognize that these misunderstandings have led to redundancy, bugs, and even dead prefabs. It turns out, the use of prefabs favors reusability, but, as their use grows in projects, it can potentially lead to a prefab hell (drawing a parallel with the infamous DLL hell). Furthermore, prefab reuse can also happen among video games, as is the case of video game sequels.

The findings of this paper can potentially benefit reuse not only in GSE but also in software engineering in general. One can argue that, in software engineering, the closest things to prefabs are the prototype pattern and the class concept itself. However, in GSE, where both the prototype pattern and classes also exist, prefabs occupy a different space. Deepening understanding on prefabs, as this paper does, may motivate other researchers to come up with new ideas to bring prefabs to the software engineering community at large.

The rest of the paper is organized as follows: Section 2 presents the related work, reviewing previous studies and contextualizing the research within the state of the art. Section 3 describes the research method, detailing the procedures, materials, and approaches used to carry out the study. Section 4 presents the results of the research derived from the study conducted. Section 5 presents the discussion. Section 6 discusses the threats to validity. Finally, Section 7 concludes the paper.

2. Related Work

Reusability is one of the most studied topics by the academic community in the context of GSE [2, 3]. Table 1 presents a comparative analysis of the works related to this topic. Three related works utilize video games from a real context in their work, but they employ a single method of data collection and do not examine a contemporary phenomenon. Video game analysis (6 works) is the most popular technique when it comes to studying reuse. Interviews (3 works) and focus groups (2 works) are the subsequent most popular techniques for studying reuse. None of the related works combine more than one technique. Moreover, none of the analyzed works mention the term or concept of prefab nor do they refer to the prototype design pattern, which is the term closest to prefab.

Ampatzoglou et al. [15] demonstrates that using design patterns improves maintainability, flexibility, and reduces complexity and coupling in game development, despite increasing the codebase size. This conclusion is supported by analyzing two open-source games, Cannon Smash and Ice Hockey Manager, using software metrics to compare versions with and without design patterns.

Kessing et al. [16] present an approach to designing semantic game worlds through a model that enriches virtual entities with diverse characteristics. Their approach promotes reusability and innovation, enabling designers to reuse and creatively modify entity semantics. To evaluate their approach they interview different game developers.

Olsson et al. [17] analyze multiple video games developed by a small video game company.They evaluate a method for detecting architectural problems in a game product line. In [18], these

[1] Prefab is the term used in the Unity engine, the most widely used game engine[12], but the concept exists in subsequent commonly used engines with other terms such as blueprint class (in the case of the Unreal engine) and scene (in the case of the Godot engine). Throughout this work, we use the term prefab because it is the most popular term in the game development community.

Table 1: Comparison of papers of the related work

Work	Year	Type of paper	Real Context	Interviews	Focus Group	Visits	Source Code Analysis	SW Phylo	Prefab
[15]	2007	Experiment	✓	✗	✗	✗	✓	✗	✗
[16]	2012	Technical	✗	✓	✗	✗	✗	✗	✗
[17]	2014	Case Study	✓	✗	✗	✗	✓	✗	✗
[18]	2015	Case Study	✗	✗	✗	✗	✓	✗	✗
[19]	2015	Survey	✓	✓	✗	✗	✗	✗	✗
[20]	2016	Technical	✗	✗	✗	✗	✓	✗	✗
[14]	2016	Survey	✓	✗	✗	✗	✗	✗	✗
[21]	2017	Technical	✗	✗	✗	✗	✓	✗	✗
[22]	2018	Case Study	✗	✗	✗	✗	✗	✗	✗
[23]	2019	Empirical	✓	✗	✗	✗	✓	✗	✗
[24]	2020	Empirical	✗	✗	✗	✗	✗	✗	✗
[12]	2021	Comparative Study	✗	✗	✗	✗	✗	✗	✗
[13]	2021	Technical	✗	✗	✗	✗	✗	✗	✗
[25]	2022	Experiment	✗	✗	✓	✗	✗	✗	✗
[26]	2023	Experiment	✗	✗	✓	✗	✗	✗	✗
[27]	2023	Survey	✓	✓	✗	✗	✗	✗	✗
This work	2024	Multiple Case Study	✓	✓	✓	✓	✓	✓	✓

same authors present a quality model based on metrics such as portability and rendering engine independence. They then apply this model to evaluate different evolutions of the MVC architecture pattern in game development.

Wang and Nordmark [19] compare game development with traditional software engineering, highlighting the differences in the life cycle, functional requirements, and the importance of usability and performance in games. They point out that game engines support reuse of code modules, but they do not specify what those modules consist of.

Ghoreshi and Haghighi [20] focus on test case generation. Their approach extracts tests incrementally from specifications, potentially improving the reusability of test cases. However, they do not study reusability beyond test cases of other game assests.

Aleem et al. [14] present the Digital Game Maturity Model (DGMM), a comprehensive framework for evaluating game development methodologies within organizations. They indicate that the better that software engineering practices are used, the better reuse will be, and they analyze the characteristics of development teams for reuse, but they do not analyze how specific reuse practices are in the context of video games.

Boaventura and Sarinho [21] present a simplified collection of game assets based on game features that enable the building of small and casual games without major modification. It emphasizes the reuse of game elements by providing a framework with minimal features and basic behaviors for game logic, which can then be adapted to multiple game platforms.

Nunnari et al. [22] discuss a software development practice for reusing motion controllers of virtual humans across different game engines. The authors propose using transpilation, which allows code written in one language to be reused in multiple runtime environments.

Khanve et al. [23] investigates the presence of JavaScript code smells in web games using the JSNose tool. The findings suggest that existing code smell detection tools are insufficient for identifying game-specific smells, highlighting the need for specialized detection tools tailored to the unique aspects of game development.

Mizutani and Kon [24] present a reference architecture designed to reduce the cost of implementing, improving, and refactoring economy mechanics in digital games. Although they stress that architecture is important for reuse, they do not explain why or how it is reused.

Politowski et al. [12] compare open source game engines from three perspectives: bibliographic, code, and human. The study reveals that game engines are not widely studied in software engineering research and present differences from traditional frameworks. They highlight that game engines and frameworks are modular platforms designed for reuse, offering standardized development processes and abstracting implementation details for easy developer input. The widespread use of game engines demonstrates their importance in modern development; although it states that the motors facilitate reuse, they do not explain why.

Geisler and Kavage [13] develop an aspect-oriented metaprogramming language designed for software reuse in video game development. The motivation to reuse drives its design, as it facilitates the translation of software artifacts across multiple game engines and projects, promoting efficiency and modularity.

Trasobares et al. [25] investigate whether Software Product Lines (SPLs) in Game Software Engineering (GSE) offer benefits similar to those in Classic Software Engineering (CSE) in the context of Model-Driven Development (MDD). They found that SPLs improve correctness but that there are no significant differences in efficiency or satisfaction. Chueca et al. [26] extended this research to the context of Code-Driven Development (CDD), finding that SPLs improved correctness, efficiency, and satisfaction in CDD paradigms.

Nardone et al. [27] identifies and categorizes 28 bad practices in video game development, validated through forum analysis and a developer survey. It highlights the importance of addressing performance and user experience issues early in the development process to improve game quality. Although video game developers mentioned prefabs in the interviews, the paper itself does not tackle prefabs at all, stressing how under-the-radar prefabs remain in the scientific community.

From the related works, we have derived three key ideas that we aim to corroborate: (1) the widespread use of game engines (understood as implementation frameworks) [12]; (2) the motivation to reuse [13]; and (3) the general characteristics of development teams for reuse [14]. Unlike previous works that propose specific frameworks or methodologies, our research provides a comprehensive analysis of the real world. By conducting a multiple case study, we analyze reuse across various contexts through different types of interviews and analyses. This

has enabled our work to shed light, for the first time in the literature, on the role that prefabs play as the fundamental unit of reuse in GSE.

3. Research method

In this section, the objectives and research question of the study are presented. The multiple case study design is also explained, as well as the procedure followed in data collection and the analysis to achieve the objective. Finally, the selection of cases, units of analysis, and subjects for the study is described.

3.1. Objective and research question

We have organized our research objective using the Goal Question Metric template for goal definition [28]. Our goal is to **analyze** software reuse in video games **for the purpose of** improving knowledge, **with respect to** elements, methods, tools and techniques **from the point of view of** professional developers, **in the context of** the video game industry.

Our research question is to determine how software reuse is adopted by video game developers in industrial settings. Specifically, we want to know what kind of software elements are reused in a video game and what techniques and tools are used.

The overarching research question can be refined into three more specific research questions, which will provide a clearer direction for our research work:

RQ1 - Do the existing findings on reuse in GSE, in terms of the technologies and tools utilized, align with real-world contexts?

RQ2 - Are there particular aspects of reuse in GSE that distinguish it from reuse in other software development contexts?

RQ3 - How does reuse in GSE relate to the specific characteristics of the video games being developed?

The character of our study is descriptive, portraying the current status of reuse in video games and also exploratory, finding out how the reuse is in video games in order to seek new insights, and generate ideas and hypotheses for new research for the improvement of software reuse.

3.2. Design, data collection, and analysis procedure

According to the classification given by [9] for case study designs, our design corresponds to that of a multiple case study consisting of two case studies, each with several embedded units of study. Figure 1 illustrates our study, including the company names and project names.

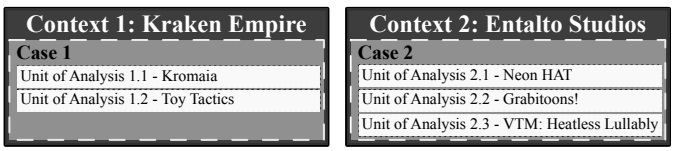

Figure 1: Mutiple case studies scheme

We divide the fieldwork of our multiple case study into three different phases, all of which are aimed at obtaining, analyzing, and extracting information about reuse in video games in an incremental and flexible way. These phases will allow us to delve into the problem from general aspects to more specific ones and to use the results and findings of each phase to refine the research in the subsequent phases. Table 2 shows the relationship between the data sources involved in the study, the data collection method used for each of them, the evidence and the type of information obtained for subsequent analysis, together with the phase assigned for its development and the research questions we intend to answer based on the evidence obtained. The three phases of the field study are described below.

PHASE I. *Capturing general information on reuse in video game development studios*

In the first phase (see left of Figure 2), the first contact is made with the video game studios. In this phase, the framework collaboration agreement is signed with each of the video game studios, the different people within the company that will participate in the case study are identified, and preliminary information about the company and the different video games to be analyzed in each of the companies is collected.

To collect initial information, a semi-structured interview was conducted with a representative subject of each studio participating in this research. In this phase, two individuals participated: one from Kraken (KR1) and another from Entalto (EN1). For these interviews, a template was designed with several sections that allowed us to collect different types of data:

- Demographic information of the interviewees and their role within the company.
- Information to characterize the company and the projects.
- Answers to questions about how reuse is performed in the developments made in each of the video game studios

The sequence of actions for conducting the interviews was as follows. First, the template together with the informed consent form was sent to the representative subject selected in each participating video game studio. Each subject autonomously completed the template and sent it back together with the signed informed consent form. Next, the initial answers provided by the subjects were reviewed, and a new set of questions was created to clarify unclear aspects, delve into specific points, or address new issues motivated by the subject's responses. Finally, a date and location were arranged with each subject for a semi-structured interview, utilizing their responses in the initial template and the newly created questions.

PHASE II. *Capturing specific information on reuse in different types of video games*

In the second phase (see center of Figure 2), phylogenetic trees were constructed to analyze the reuse of content across video games produced by participating video game companies. Phylogenetics is a field in biology that uncovers ancestry relationships among individuals, classifies them, and constructs an evolutionary tree that illustrates their evolutionary phenomena[29, 30]. This field involves studying the relationships among taxa and hypothesizing about their evolutionary history. Taxa are groups of organisms defined by shared characteristics or attributes, such as a species being a taxon. Phylogenetics can reveal hidden connections among individuals

Table 2: Relation of data sources, methods of data collection, evidences obtained, and type of information analyzed.

Data sources	Collection methods	Evidences	Type of information	Phase	Research Questions
Lead developers *Two subjects, one from each company*	Questionnaires	Responses to Phase I questionnaires	Quantitative and qualitative information on each video game company, subjects and reuse in GSE	Phase I	**RQ1, RQ2**
	Interviews	Researchers' notes and interview transcripts	Qualitative information and experiences on reuse in GSE	Phase I	**RQ1, RQ2**
Videogame projects *Five projects , three from one company and two from another*	Archival Analysis	Videogames source code and Phylogenetic trees	Quantitative and qualitative information on reuse in a video game project	Phase II	**RQ2, RQ3**
Lead developers *Two subjects, one from each company*	Questionnaires	Responses to Phase II questionnaires	Quantitative and qualitative information on each video game, subjects and reuse in a videogame project	Phase II	**RQ2, RQ3**
Developers *Two subjects, one from each company*	Focus group	Researchers' notes	Qualitative information and experiences on reuse in a videogame project	Phase III	**RQ2, RQ3**
Visit *Four subjects, all from the same company*	Direct Observation	Observation sheets and researchers' notes	Qualitative information and experiences on reuse in a videogame project	Phase III	**RQ1, RQ2, RQ3**

that have not yet been discovered. Recently, this approach has been adapted for use in software, where software components (e.g., methods or classes) serve as taxa for phylogenetic inference [11, 31].

Charles Darwin suggested that life evolves through descent from common ancestors, and he introduced the metaphor of the "tree of life" [32] to convey this idea. Inspired by this analogy, Baum et al. [33] initiated the Tree-Thinking movement, asserting that anyone, even without knowledge of phylogenetics or biology, could interpret these trees and use them for organizing and classifying information. Consequently, the most common representation of phylogenetic inference results is the phylogenetic tree, or phylogram. This undirected graph features nodes (representing taxa) and links (representing ancestral relationships among taxa), with edge lengths indicating the differences among taxa [34].

The term "evolution" in this work is derived from concepts presented by Baum and Smith [33]. In this context, "evolution" does not necessarily refer to the traditional time-related sense but instead focuses on the paths or relationships between entities that are not explicit or directly tied to chronology. This perspective suggests that prefabs may exhibit latent trends or drifts based on shared or differentiating characteristics, revealing implicit relationships that might otherwise remain hidden. Thus, in the phylogenetic tree presented here, the trajectory between two prefabs represents an inferred progression, which is neither necessarily linear nor linked to direct temporal succession but is determined by their evolutionary relationships.

We sent the participating subjects the questionnaire via email along with the phylogenetic tree of the video games to be analyzed, the informed consent, and contact information to resolve any doubts or questions about the questionnaire. The subjects sent us their responses and signed informed consents, and we analyzed the evidence obtained from each video game.

This phase involved analyzing the five unity of analysis, one for each video game studied: ToyTactics, Kromaia, VTM, NeonHAT, and Grabitoons!. Within each context, one or two developers participated, depending on the project. These participants, who had worked on the development of the respective video game, answered questions about specific reuse practices and analyzed the phylogenetic tree to identify patterns of reuse. Each video game was analyzed by at least one developer, and at least one of the games from each company was analyzed by two different developers, with different roles within the project. Specifically, in the context of ToyTactics, EN1 and EN2 participated; in Kromaia, EN1; in VTM, KR1 and KR2; in NeonHAT, KR1; and in Grabitoons!, KR2.

To collect information, we designed a questionnaire with several sections, aimed at gathering insights from the developers. The sections included:

- Demographic data about the participants, their roles in the company (Only for new participants), and their involvement in the analyzed video game.
- Responses to specific questions about reuse within the analyzed video game.
- An analysis of the phylogenetic tree provided for the video game, with questions focused on the patterns of reuse inferred from the tree.

The questionnaire, along with the phylogenetic tree of each video game, the informed consent form, and contact information for resolving doubts, was sent to the participating developers via email. The developers completed the questionnaire and returned it alongside the signed consent forms. The responses were then analyzed to deepen our understanding of reuse practices and to identify patterns observed in the phylogenetic trees of the analyzed video games.

PHASE III. *Sharing preliminary results and the formulation of final conclusions.*

In the third phase of the fieldwork (see right of Figure 2), after analyzing all of the evidence and sharing it with the research team members, a preliminary summary of the results was

prepared, and two focus groups were organized—one for each company analyzed. The subjects who had completed the questionnaire in the previous phase also attended each of the focus groups.

The use of Focus Groups instead of interviews in this third phase of the study facilitated the discovery of new points of view due to the different roles, experiences or backgrounds of the different participants in the study. It helped us to confirm or reject facts that would have been hidden in an individual interview, and to go deeper into certain aspects, since a focus group allows us to discuss different and complementary opinions. The focus group was also considered beneficial for the participants, as they were able to compare their way of working. Moreover, in terms of research costs, it is more cost-effective than individual interviews.

For each focus group, a list of specific questions was prepared to address common and distinct aspects of reuse in the different video games analyzed. These questions also included inquiries on reuse practices arising from the analysis of the evidence obtained in previous phases.

In addition, this phase included a visit to the facilities of Entalto, where four employees were working, two of whom had participated in the focus group. This visit allowed us to gain a deeper understanding of their work dynamics and the tools they use on a daily basis. Before the visit, we devised an observation sheet to systematically document insights during the visit.

In summary, we have collected the following data: two responses to the general questionnaire, seven responses to the specific questionnaire (including one for Kromaia, two for Toy Tactics, one for Grabitoons!, two for VTM, and one for NeonHAT), two focus groups conducted, and one visit completed.

The replication package of this work can be found at `https://doi.org/10.5281/zenodo.14591667`. It contains the following: the different framework agreement and informed consent models used following the guidelines of our university ethics committee, the templates and lists of questions used in the interviews and the focus group, the observation sheet used in the visit, the tool that creates the phylogenetic trees from the source code [11], and the phylogenetic trees.

3.3. Cases and subject selection

In this work, we examine two contexts (companies): Kraken Empire and Entalto Studios. Table 3 shows a comparative overview of these two companies. Kraken Empire was founded in 2011. It currently has four developers, but it has had up to twelve at its peak. Entalto Studios was founded in 2021 and currently operates with a team of six developers, but it has had a maximum of eight. Both companies also engage minimal additional roles, indicating a lean development structure that is focused on efficiency. Both companies have developed the same number of their own games (3), although Entalto Studios has participated in a larger number of video games (13) compared to Kraken Empire (3). This is because Entalto Studios also works for other companies, doing ports to other platforms, and producing and post-producing video games, whereas Kraken Empire solely develops its own games. In terms of programming languages, both companies use C++ and C#. However, Entalto Studios also uses HLSL and GLSL, which are shader languages for OpenGL with regard to game engines, both companies use Unity. Additionally, Entalto Studios utilizes Unreal and GameMaker, Kraken Empire uses its own game engine that uses different libraries like OGRE 3D. Both companies develop games for a variety of console platforms and PC. Entalto Studios also develops games for VR devices, while Kraken Empire develops games for mobile devices. Another difference is that Entalto Studios operates in an on-site setting, whereas Kraken Empire functions online.

Table 3: Comparison of companies

	Kraken Empire	**Entalto Studios**
Year of Foundation	2011	2021
Years of service	13	3
Developers (Now)	4	6
Developers (Max)	12	8
Type of Location	Online	On-Site
Game Engine	Unity, Own Engine	Unity, Unreal, GameMaker
Programming language	C++, C#	C++, C#, HLSL, GLSL
Own Games	3	3
Games Developed	3	13
Platform	PlayStation, Xbox, Nintendo Switch, PC, Mobile Devices	PlayStation, Xbox, Nintendo Switch, PC, VR

We argue that these companies to be relevant since the selected companies use the most widespread engines and publish their games on the most popular platforms. Both companies belong to the Indie sector within video games since, from 2018 to 2023, indie game publishers made up about 99% of the releases on gaming platforms [35]. Indie games represent a continuously growing market challenging the dominance of AAA titles with innovative and creative approaches [36]. Furthermore, the range of game engines, programming languages, and platforms of these companies also makes these companies relevant.

Table 4 shows a comparative analysis of five games from Kraken Empire and Entalto Games.With the exception of Kromaia, the Unity engine was utilized in the development of four of the five video games analyzed. The programming languages employed are analogous across all projects.However, there are differences in development duration and project scale. Kraken Empire's "Kromaia" and "Toy Tactics" had longer development cycles, taking 48 and 72 months, respectively, compared to Entalto Games' much shorter timelines. For instance, "Grabitoons!" took twelve months, "NeonHAT" took ten months, and "VTM: Heartless Lullaby" was developed in just two months. This indicates that Kraken Empire invests in longer-term, more elaborate projects, while Entalto Games focuses on quicker, more agile development cycles. In terms of project scale, Kraken Empire's games involve a higher number of lines of code, particularly in "Toy Tactics". In contrast, Entalto Games' projects, though varied in complexity, generally involve fewer lines of code. The platforms targeted by each company for their games are also different. Kraken Em-

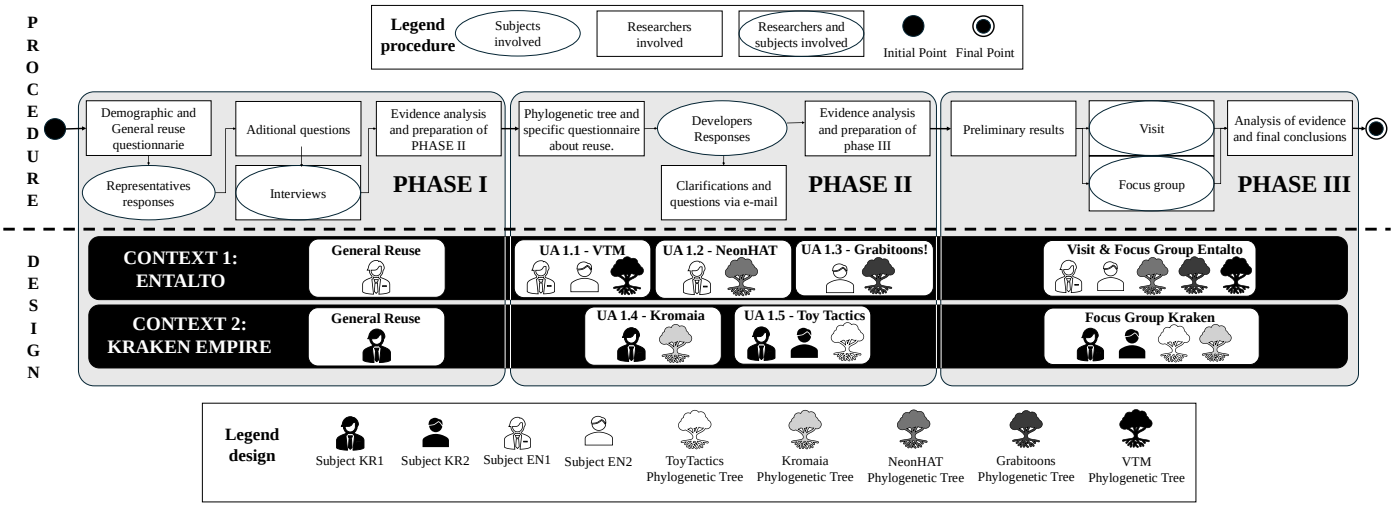

Figure 2: Design and analysis procedure

Table 4: Comparison of games

	Kraken Empire		Entalto Games		
	Kromaia	**Toy Tactics**	**NeonHAT**	**Grabitoons!**	**VTM: Heatless Lullaby**
Development Duration	48 months	72 months	10 months	12 months	2 months
Year	2014	2024	2021	2022	2022
Engine Used	Own	Unity	Unity	Unity	Unity
Developers	2	12	5	5	7
Other roles	1	4	1	1	1
Game Duration	10 hours	20 hours	4 hours	9 hours	1 hour
Lines of Code	145,000	128,000	89,138	46,053	20,758
Game Genre	Spatial Simulator	Real Time Strategy	VR Racing	Party-Game	Narrative CRPG
Programming Language	C++	C#	C#, HLSL	C#, HLSL	C#, HLSL
Plataform	PC, PlayStation	PC, PlayStation, Xbox	PSVR, HTC Vive, Quest	PC, Switch	PC
URL	bit.ly/vg_kr	bit.ly/vg_ty	bit.ly/vg_nh	bit.ly/vg_gb	bit.ly/vg_vm

pire focuses on traditional gaming consoles like PlayStation and Xbox, alongside PC. Meanwhile, Entalto Games extends its reach to VR platforms such as PSVR, HTC Vive, and Quest, indicating a strategic focus on emerging VR technologies and immersive experiences. This distinction highlights Kraken Empire's emphasis on conventional gaming systems versus Entalto Games' exploration of innovative, cutting-edge platforms.

We consider these video games to be relevant since each video game is listed from a different genre, showcasing the diversity in both studios' portfolios. Kraken Empire's "Kromaia" is a Spatial Simulator, while "Toy Tactics" is a Real-Time Strategy game. Entalto Games' "Grabitoons!" is categorized as a Party-Game, "NeonHAT" as a VR Racing game, and "VTM: Heartless Lullaby" as a Narrative CRPG. The variety of genres represented in this sample reflects the diversity of factors that can influence the development of a game. In addition, the video games selected span several of the most prevalent genres on personal computers, video game consoles, and mobile platforms, including first-person shooters, strategy, racing, or simulation games [37].

Table 5 shows a comparison of the 4 participants in this study, two from Kraken Empire (KR1 and KR2) and two from Entalto Studios (EN1 and EN2). Subjects KR1 and EN1 participated in all phases, whereas subjects KR2 and EN2 were involved only in PHASE II and III the study. There are differences in age and experience, with KR1 being the most experienced, with 20 years in software development and 18 years in video game development. In contrast, KR2, EN1, and EN2 are young and less experienced, with only 3, 4 and 6 years of experience respectively. Their roles in the developments also vary. KR1 has a leadership position in the company, and he is a senior developer with the role of leader developer in the projects analyzed, while KR2 is a junior developer. EN1 and EN2 are both developers in Entalto; however, EN1 assumed the role of leader developer in the analyzed video games, whereas EN2 was more focused on specific aspects of the development process.

We consider these professionals to be relevant since the par-

Table 5: Comparison of participants

	Kraken Empire		Entalto Studios	
	KR1	**KR2**	**EN1**	**EN2**
Age	40-50	25-30	25-30	25-30
Role	Director and project leader	Developer	Manager leader and developer	Developer
Years developing software	20	1	4	6
Years developing videogames	18	3	4	6
Experience in programming languages	C, C++, C#, ASM-8086, Java, Python, Ada	C, C++, C#, Java	C#, C++, HLSL, GLSL	GML, C++, Java, C#, PHP, JS
Frequent programming languages	C#	C#, Java	C#	C#
Experience in game engines	Unity, Unreal	Unity, Unreal	Unity, Unreal, GameMaker	Unity,Unreal, GameMaker
Frequent game engines	Unity	Unity	Unity	Unity

ticipating subjects were chosen, from the professionals available in each company, in order to represent different development experiences and different roles within a project. This provides different points of view in the study. The age of the participants can also be considered to represent the youth of the video game industry since almost half of video game developers are under 35 and more than 3/4 are under 45 years old [38].

4. Results

In this section, we present the findings on reuse in game development after analyzing the evidence collected during the different phases of the study. The results on reuse in video game development are presented below, with the statements expressed by the participants, indicating whether each statement was obtained from the questionnaire Q "..." or obtained through an interview I "...". We include statements following the recommendations of the methodological frameworks [9]. The participants' statements appear in italics.

First, the general results are derived from the interviews and questions used during the first phase of our research. In this phase, the prefab appears as the fundamental element or unit of reuse in the context of video games. This leads us to present the specific results of our second and third phases of research, where we delve deeper into the concept of 'prefab' or unit of reuse, and use it to analyze the reuse that takes place in the development of a video game.

4.1. Reuse in the context of video game studios

In order to present an overview of the reuse that occurs in the context of video game development, we analyzed the responses and observations of the participants during PHASE I of this research and organized them around the three general results presented below.

General result 1. *Elements, techniques, and tools for reuse*

The elements most commonly reused in video game development include classes, scripts, class systems, and particularly prefabs. To facilitate reuse, developers primarily rely on the game engine editor, while direct script reuse is typically managed through the IDE. Both studios also incorporate design patterns, with the MVC pattern being a notable example. Additionally, they often develop custom tools for reuse, such as bespoke level editors, to streamline designers' workflows. This insight addresses the question: "What techniques and tools are used or not used for reuse?"

I "*It's better to reuse prefabs because there are so many of them in a video game. This can affect both development times and performance.*"

Q "*Yes, for everything that's scripts, reference architectures or patterns and good practices are used. For different assets, most of them prefabs, the engine itself or its own tools are used.* "

Q "*We use the Model-View-Controller (MVC) pattern and try to encapsulate as much as possible.We also developed our own level editor so that the designer could create new ones without needing to know the implementation details. During the pre-production phase, the designer made levels by hand. Once we'd developed the race/level editor, the designer was able to design more races/levels without making as many mistakes.*"

Q "*In the IDE, you do software engineering and architecture. In the editor, you can use the engine's features (dragging and dropping objects into a scene).*"

The selection of an engine is a matter of resources and the specific requirements of the development project. It should be noted that not all engines are free of charge, and that not all of them provide the necessary utilities. Furthermore, the knowledge of the team is a fundamental factor in this decision. The aspect of the engine that facilitates reuse is Unity's prefab system. Nevertheless, all engines facilitate reuse in a comparable manner.

I "*The prefab system is really useful for any Unity project. It lets you put all your data and behaviours in one place. The prefab system lets you structure things in a centralised way, and then share that structure between different elements. You can then just change the parameters, like attributes or behaviour.*"

In this study, no generative AIs or automatic code generation were employed for any type element of prefab, neither as a foundational base for generating ideas nor for refining or detailing them. The participating subjects emphasized their reluctance to adopt such approaches, citing a lack of identified advantages and expressing limited positive regard for their use.

I "*It is not at the level of quality that a human can offer (it is not up to par in code or art). Plus, it doesn't look very good.*"

General result 2. *Motivation to reuse*

Reuse offers several advantages, such as increased agility, faster and more efficient game development, improved product quality, reduced technical debt, and easier maintenance of the video game. However, reuse also has its drawbacks, including the time and effort required to design or build reuse techniques, when you are going to use only once time and implement reusable components, as well as the risk of propagating unintended errors. To get these advantages and disadvantages, we asked: "Why do you reuse? or What are the advantages and disadvantages of reuse?"

Q "*The advantages of this approach are numerous. Firstly, it is much more agile and straightforward to develop the game. Secondly,*

it reduces the technical debt that accumulates in the latter stages of development, when it is necessary to debug more intensively. The only disadvantage is that at the outset, development is slower due to the necessity of designing and implementing the various reuse techniques. This results in critical junctures during the development process where it is unfeasible to proceed and an ad hoc solution must be implemented, which may result in a faster delivery date in the short term."

General result 3. *The characteristics of the development team for reuse*

In order to capitalize on the advantages of reuse, it is essential that the developers on the project team, regardless of their role, possess a fundamental understanding of software engineering. The concepts of inheritance, composition, and design patterns are of particular importance. The team should be well-versed in the work environment and practices that facilitate reuse. They should be familiar with the concept of a prefab and its appropriate utilization. The more experience the team possesses, the more effectively reuse can be executed. To explore this further, we asked the question:" Does the development team need to have some specific qualities in order for the game software that employs reuse to maintain quality, reliability, and portability standards?"

Q "*They need to know the work environment and practices that encourage reuse. They need to know what a prefab is and how to use it correctly. If you are a programmer you need to know patterns and architectures needed for code development.*"

Q "*Solid knowledge of software engineering, especially inheritance, composition and design patterns.*"

I "*I think as we have gained more experience, we make more use of reuse as we see the beneficial implications it has in the long run.*"

I "*A larger number of senior developers allows more control.*"

The results of our PHASE I confirm the claims of previous work related to: (1) the widespread use of game engines (understood as implementation frameworks) [12]; (2) the motivation to reuse [13]; and (3) the general characteristics of development teams for reuse [14]. Moreover, for the first time in the scientific literature, our PHASE I reveals the use of prefabs as fundamental units of reuse. Therefore, PHASE II and III delve deeper into prefabs with the following specific results.

4.2. Fundamental unit of reuse: Prefab

Specific result 1. *Better understanding of Prefabs as fundamental units of reuse*

In Unity, a prefab is a reusable object representing a GameObject. A GameObject is the building block, representing various elements like characters, props, scenery, cameras, waypoints, and more. The functionality of a GameObject is defined by the components attached to it. Unity prefabs serve as a template that allows you to save the configuration of a GameObject in your scene as an asset in your project. When you create a Unity prefab, you capture a specific setup of a GameObject, including its components, properties, and child GameObjects. Unity prefabs act as reusable assets, enabling you to create new instances in your scenes easily and maintain consistency across multiple copies. Any changes made to the original Unity prefab are automatically reflected across all instances. Unity prefabs also allow for nesting, where you can embed Unity prefabs within other Unity prefabs to create complex hierarchies. This feature makes it easy to edit different levels of the hierarchy while maintaining flexibility for unique settings in individual prefab instances if needed. Essentially, prefabs in Unity are concepts for efficient and consistent GameObject management, allowing developers to streamline the creation and maintenance of GameObjects. To get these ideas, we have asked: "How would you define a prefab as a minimum unit of reuse?", "Are prefabs the components of a video game? Are prefabs the characteristics of a video game?

Q "*Prefab is an artifact that centralizes a behavior or data usage and then in the game code targets to it.*"

I "*Prefab is the skeleton of GameObject*"

A Unity prefab is like a template for creating GameObject instances, similar to a class or prototype pattern in programming, but more flexible. It can include scripts, colliders, particle systems, and child GameObjects, making it a powerful tool in game development. One notable aspect of prefabs is that they handle game objects and scripts differently. If you modify a prefab's GameObject, the change affects only in that instance. However, changes to a script in a prefab affect all prefabs using that script, as the script can be shared across them. In this part the question was: "How does the prefab implementation differ from the prototype pattern or class?"

Q "*The prefab is an object with several components already added that acts as a base to create similar or similar objects. If we compare it to OOP, a prefab would be a class. Once you have defined what an "Animal" class is, you are not going to implement again the variables float hunger, maxVelocity, etc., and all their functionalities, like Eat() when you create a "Cat" class; but you will reuse the implementation of the "Animal" class that already has all that as a base, inheriting from it. The same thing happens with prefabs: if all the enemies are going to share 80% of their functionality and components, it is likely that an "Enemy" prefab will be created from which each of the enemies will be created with their own particularities, but already having that base.*"

I "*Prefab is more versatile than a class or the prototype pattern; it is more integrated in unity. It also has more potential and gives you predefined components that the prefab can have and the class does not.*"

In other game engines, there are similar concepts but with different names and slightly varied functionalities. In Unreal Engine, the equivalent of Unity prefabs is called Blueprint Classes [39]. Blueprint Classes are visual scripting assets that define the behavior and properties of objects. They allow for the creation of complex interactions without directly writing code, making them especially useful for game logic, AI, and interactions between objects. While both Unity prefabs and Blueprint Classes provide a way to create reusable templates for GameObjects, Blueprint Classes are more focused on visual scripting and can encapsulate complex logic beyond just GameObject setup. The question was: "Whats is the name of Prefab in other engines?"

I "*I think in Unreal, prefab is blueprint*"

In the Godot Engine, the closest counterpart to Unity prefabs

is the concept of scenes [40]. A scene in Godot represents a collection of nodes, similar to GameObjects, that one organized hierarchically. Scenes can be saved as separate files and reused across different parts of your game or even in other projects. While scenes in Godot serve as templates for creating instances like prefabs, they are more comprehensive, encompassing not only GameObject setup but also the entire hierarchy of nodes and their interactions. This makes scenes in Godot a versatile concept for structuring and reusing complex game elements across different projects.

I "*Prefabs in Godot, for example, are called scenes, but scenes in Unity are something else.*"

In the case of Kromaia, the developers utilized their proprietary engine to create prefabs based on XML files. These XML files contained various components like tags such as "Hull," "Link," and "Gun," each with a series of modifiable attributes. This allowed for a high degree of customization and flexibility in the game's design, as each component could be tailored to specific needs and functions.

Similar to other game engines, Kromaia's engine featured a reusable unit concept, though with a different name and slight variations. These reusable units help to reuse small game elements, speeding up the development process and improving the consistency and coherence of game assets. We call this concept prefab because it is the name most commonly used among the developers interviewed.

Specific result 2. *Reusing Prefabs between video games*

In video game development, there are cases where elements can be reused between titles of the same franchise, such as in the case of *Kromaia X*, the sequel to *Kromaia* adapted for VR, this practice remains limited and is beyond the scope of this work. In these specific cases, the reuse of prefabs proves useful to maintain narrative and visual coherence within the saga.

However, our findings reveal that reuse between standalone games tends to be very low. When we asked developers, "How much have you reused from other projects (own, external sources, or the store)?" their responses indicated reuse rates ranging from only 2% to 5%. This is largely because the industry prioritizes delivering unique and exclusive experiences, focusing on the development of assets and mechanics that distinguish each title from its competitors. These exclusivities are seen as key selling points, strengthening brand identity and attracting dedicated fan bases. This emphasis on innovation and uniqueness significantly restricts the extent to which prefabs from one game can be reused in another. Additionally, factors such as the unique artistic styles, narrative structures, and gameplay mechanics of each game further complicate the reuse of elements from previous projects. The question to know it is:"Why is reuse between video games so low?"

I "*In our games, we can not reuse almost anything because we have created different games of different genres, different mechanics, and different artistic styles.*"*(Entalto)*

I "*It was not possible to reuse elements in other projects because you do mechanics or components and they are so specific and exclusive that they are impossible to reuse.*"*(Toy Tactics)*

With the results obtained, our focus will shift to reuse within a single game, leveraging phylogenetics as a tool to visualize and analyze the relationships and reuse patterns of its components.

Specific result 3. *First insights on prefab evolution*

Figure 3 shows the phylogenetic trees of each of the video games and the number of prefabs in each case. The trees are the result of applying the Phylogenix tool [11] to the source code of each video game. The trees can be found in the replication package for this study. The trees conform to the widespread standard of phylogenetic trees and can be explored with standard visualization tools such as iTOL [41] (the url to use the visualization tool is also in the replication package).

When we asked of the game developers whether they could interpret the reuse in their video game from the phylogenetic trees, they provided affirmative responses and their interpretations.

Q "*I see that some very similar items are close to each other, such as HitSaveRun/1pointBox, HitSaveRun/2pointBox, and HitSaveRun/3pointBox, which are objects that give points on the same level. I also see that the VFX that are used in the final game are close together*" *(Grabitoons!)* Figure 4 depicts what the developer has specified.

Q "*Most of the information makes sense, such as similar objects being together in the tree (e.g., the four types of Dancers/Dancer there are). The nine types of Chapter2Disco/Walls that are together above and share many similar components.*" *(VTM)* Figure 5 illustrates what has been indicated by the developers.

Q "*Objects are close to each other that have a single based on another, such as Common/Dog and HitSaveRun/HSR_Dog. The latter is a variation of the former.*" *(Grabitoons!)* Figure 6 shows what the developer indicates.

In addition, we asked with the developers about the potential meaning of the shape of the philogenetic tree. For Entalto developers, a considerable number of highly similar prefabs was indicative of suboptimal reuse management, as they could likely be unified into a smaller number. To illustrate this, they compared Neon HAT with Grabitoons!. Both games have the same size, but in Grabitoons! experience had made them perform better reuse management than in 'Neon HAT'. For this reason, they were not surprised that the phylogenetic tree of 'Neon HAT' was more elongated and with larger clusters compared to the tree of Grabitoons or VTM. However, for Kraken Empire developers, the size and shape of the clusters are not solely dependent on reuse management; rather, they are influenced by the nature of the game content. In both of the Kraken Empire games analyzed, the quality of the reuse management was deemed satisfactory. The observed differences in tree sizes and shapes may be attributed to variability and size. In Kromaia, the objects that appear in the game are distinct from one another, with few objects exhibiting similarities, resulting in small clusters and an elongated tree. Conversely, in Toy Tactics, the combat units are comprised of numerous items with great similarity, resulting in larger clusters.

Q "*Since not much reuse is done in this game, I think it's wise to have two kilometer-long branches, implying that they could be reduced.*" *(Neon HAT)*Figure 7 represents the information provided by the developer.

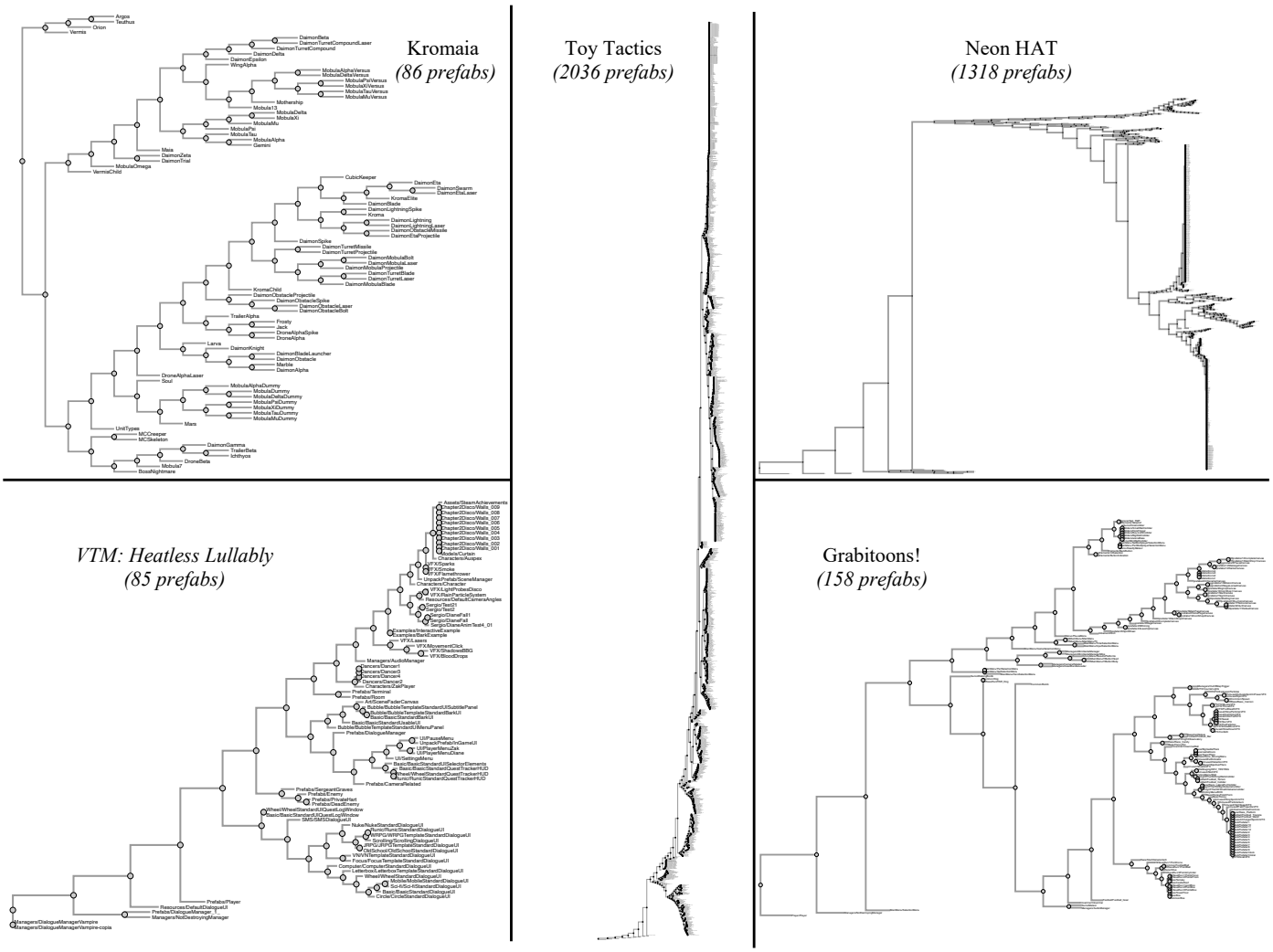

Figure 3: Phylogenetic trees of each video game analyzed with the corresponding number of prefabs

Specific result 4. *Risks of reusing prefabs*

Nevertheless, our work also reveals a lack of understanding by the developers of some reuse relationships among the prefabs. The developers stated that their understanding of these relationships did not correspond to what they saw in the trees. However, after inspecting the code, they recognized that it was their understanding that was incorrect. Neither in the projects with about 100 prefabs nor in those with more than 1000 prefabs did any developer have the full prefab reuse relationships in the head. One possible explanation given by the developers is that prefabs are developed in projects that can last for years and are built by multiple different developers.

Analysis of the trees also reveals that developers recognize that their misunderstandings of reuse in prefabs have led to redundancy and even bugs. The developers insisted that prefabs are not usually built by a single developer, but are built by several developers who may even have heterogeneous profiles such as dedicated programmers as well as artists with programming skills, known as technical artists. It turns out, prefabs have encouraged reuse in the development of video games, but they also come with potential problems of redundancy or bugs.

I "*However, I see relationships that I don't quite understand, like Sumo/Meteor and Managers/AudioManager, probably because they share structure even though they are used in the game for different things.*" *(Grabitoons!)*

I "*For a developer just getting into the middle of a project, it can help to understand the project structure because it shows relationships that the project folders do not. One thing that can help directly is to eliminate redundancy in the prefabs. In this example tree, you see, for example, the dancers that can surely be reduced to a single prefab.*" *(VTM)*

I "*The tree gives me information that I didn't have. I remembered that we did it with a few errors that caused a lot of technical debt in the last stages of development, and, with the tree, it becomes more evident what the problem is. The information given by the tree can be used to fix reuse problems.*" *(Neon HAT)*

Q "*The tree has given me more information than I thought it would. For example, we have many more VFX than I thought we would have to eliminate because they are not used in the final game, which appear close to each other (Unused/...).*" *(Grabitoons!)*

Another problem recognized by the developers was the prob-

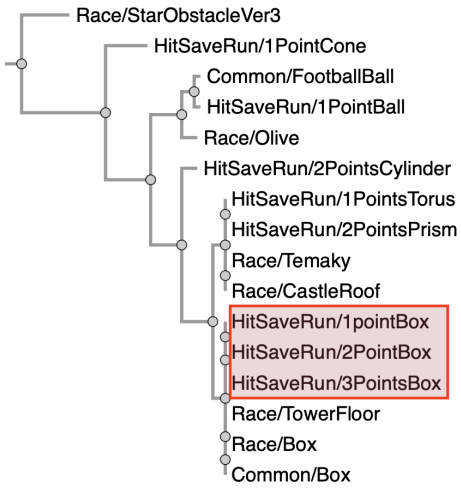

Figure 4: Observation highlighted in Grabitoons! phylogenetic tree, as indicated by the developers

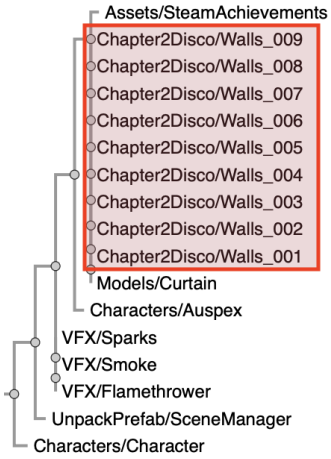

Figure 5: Observation highlighted in VTM phylogenetic tree, as indicated by the developers

lem of prefabs that are present in the video game but are not used. The developers stated that this occurs when they create prefabs that anticipate future functionality that never materializes and when the game design evolves and prefabs that were used are no longer used. The analysis of the trees revealed that the games had prefabs that we can call dead. These dead prefabs add unnecessary complexity and maintenance overhead.

I "*We did a lot of mechanics that then we didn't use. One of these was a portal system to connect different levels*" *(Kromaia)*

I "*The VFX that are not used in the final game are set apart.*" *(Grabitoons!)*

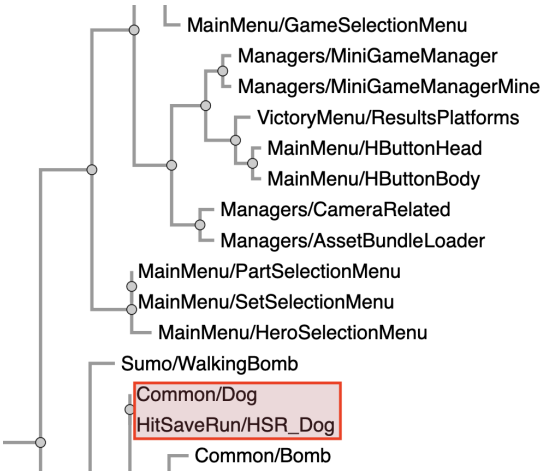

Figure 6: Observation highlighted in Grabitoons! phylogenetic tree, as indicated by the developers

The results of our PHASE II and PHASE III offer a better understanding of Prefabs as fundamental units of reuse: (1) they occupy their own reuse space compared to classes or the prototype pattern; and (2) the concept is not unique to a specific engine, but is found in other engines with other terms. For the first time, our results show how prefabs evolve and how developers can use that evolution to gain insights about reuse through the visualization of phylogenetic trees. Furthemore, our results reveal potential risks of prefab reuse related to redundancy, bugs, and even dead prefabs. Additionally, our results identify the opportunities of prefab reuse between video games.

5. Discussion

In order to analyse reuse in GSE and to determine how software reuse is adopted by game developers in industrial environments, we conducted a case study that allowed us to answer three research questions. In this section, we present how the results obtained help us to answer the research questions of this work, by improving the existing knowledge about reuse in GSE.

RQ1- Do the existing findings on reuse in GSE, in terms of the technologies and tools utilized, align with real-world contexts?

The results of the first phase of our study confirm the statements of previous work related to the widespread use of game engines, the motivation for reuse or the general characteristics of development teams for reuse, which leads us to answer our first research question in the affirmative.

RQ2- Do the existing findings on reuse in GSE, in terms of the technologies and tools utilized, align with real-world contexts?

During the first phase of our study we also discovered that in the context of videogames there is a 'fundamental unit of reuse' linked to the different development engines which in this work

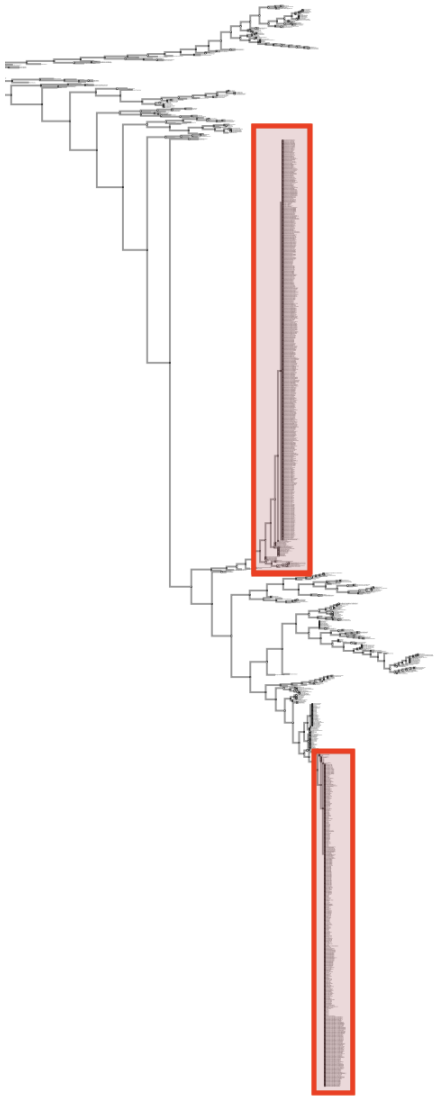

Figure 7: Observation highlighted in NeonHAT phylogenetic tree, as indicated by the developer's claim

we have called prefab and which has not been analysed in previous works. Although prefabs are already common practice in the industry, this is the first time they have been used to analyse reuse in a videogame. Prefabs are very similar to classes, as they both serve as reusable templates; however, there are key differences between the two that make prefabs uniquely suited to the context of video game development.

On the one hand, prefabs are specifically designed for game objects and assets in game development engines. They allow developers to create and store reusable elements such as characters, environments, or items that can be easily instantiated or modified across different parts of a game. Prefabs encapsulate both the visual and functional components of a game object, including its appearance, animations, and interactions with the environment, making them a key element in creating dynamic and scalable game worlds.

On the other hand, classes are a core concept in object-oriented programming (OOP) and are used to define the structure for creating objects that encapsulate both data (attributes) and behavior (methods). Classes are typically more abstract and not tied to any specific game engine or asset type. They are used to model behavior and functionality in the software, such as defining how an object responds to user input, interacts with other objects, or processes data. Although classes enable code reuse by allowing instances of the class (objects) to inherit the behavior defined in the class, they do not directly manage the assets or visual components that are fundamental in game development. Thus, the primary distinction lies in their application: prefabs are used for managing and reusing game-specific assets and objects, while classes focus on structuring the logic and behavior within the software."

The second phase of this study has helped us to better describe these elements and to better understand their relevance. Prefabs occupy their own reuse space compared to classes or the prototype pattern.

Another particular aspect that our results reveal is that reuse between different videogames is not a common practice, since the search for distinctive elements in a videogame means that a large part of the development cannot be used in new projects.

The existence of a fundamental unit of reuse and the fact that reuse between different videogames is not a common practice, support the relevance of focusing the study of reuse in GSE on reuse within the same videogame, and of basing this study on the relationships that may exist between the prefabs that compose the videogame.

RQ3 - How does reuse in GSE relate to the specific characteristics of the video games being developed?

The study of the reuse of prefabs in video games with different characteristics through phylogenetic trees, carried out in the second and third phases of the work, reveals that the reuse in a video game does not only depend on the size of the video game, but also on other factors such as the type of game, the degree of similarity between the objects that make up the game, design choices or the good or bad management of reuse done by the development team.

In addition to answering the research questions that motivated our study, this work has allowed us to identify the opportunities of prefab reuse between video games of the same saga or the versions of a video game for different platforms. We have also highlighted that the use of phylogenetic trees to study in-game prefab relationships can be a useful tool for developers.

However, prefab reuse is not without its challenges. During development, poorly managed prefab reuse can result in prefab hell, where the relationships between prefabs become overly complex and difficult to maintain. This phenomenon mirrors DLL hell in traditional software development, where mismanaged dependencies between dynamic libraries led to version conflicts and runtime failures. In the case of prefabs, issues such as element redundancy, bug detection and obsolete elements can emerge, undermining the benefits of reuse. By using tools like phylogenetic trees to visualize prefab evolution, developers can not only identify opportunities for efficient reuse but also detect and address these potential pitfalls early in the

development process, ensuring that prefab reuse remains an effective strategy. We hope that these ideas and hypotheses can be applied to improve reuse in GSE and software reuse in general.

6. Threats to validity

As in other case studies, the main threat to the validity of our work lies in its generalizability. The inherent complexity of video game development projects, and the limited scope of our cases study, in which only four video game developers actively participated, may limit the generalizability of our findings. To address this limitation, our results could be validated through a more expansive and diverse set of case studies. In this section, we describe the methods we have employed to increase the validity of our research. These methods are based on an analysis of the main categories for descriptive and exploratory case studies, namely construct validity, external validity, and reliability [10, 9].

Construct validity refers to the accuracy with which the measures of the case study reflect the concepts being studied. *Triangulation*: The empirical evidence was gathered from a variety of evidences, including questionnaires and forms completed by the interviewees, interview recordings and transcriptions, different researchers' notes and observations, or phylogenetic trees. As proposed by Yin [9], the use of multiple measures of the same phenomenon enhances the validity, reliability, and credibility of the research. In addition, three different researchers participated in the interviews and focus groups. One of them was in charge of managing the information and conducting the interviews, another one was in charge of recording and transcribing when necessary, and another one was in charge of controlling the interview protocol, including aspects that were not clear. They all took notes and stimulated the discussions. *Peer debriefing*: The work has been carried out by a team of researchers, lowering the risk of being biased by one researcher. After each phase, the researchers shared their observations and agreed on the conclusions and the next steps of the research. *Member checking*: The final conclusion and dissemination package was distributed to the participants for their review and endorsement of the conclusions and information to be disseminated. *Prolonged involvement*: The study spanned approximately ten weeks, divided into three phases. The first phase lasted five weeks, the second four weeks, and the third three weeks. Throughout this period, researchers maintained contact with the participants through various means, ensuring that they could comprehend the terminology used in the study and that the participants had sufficient time to provide data.

External validity refers to the ability to generalize the results to other cases and how to do it. To improve the generability of this work, we have opted for a multiple case study design, in which we have selected two different context that could represent small indie video game companies since, from 2018 to 2023, indie game publishers made up about 99% of the releases on gaming platforms[35]. Also, five units of analysis have been used, with the objective of covering a broad spectrum of games made in such contexts. Finally, we chose for the interviews subjects representative of different groups within the industry, only four subjects actively participated in our research.

Reliability refers to the extent to which the study operations, data collection, and analysis procedures can be repeated with consistent results. The application of *triangulation*, *peer debriefing*, and *member checking* also has a positive impact on the reliability of the work. Furthermore, a replication package has been prepared that, in conjunction with the descriptions of the design, analysis, and protocols presented in this paper, can be utilized to replicate the study in other contexts. These contexts can include in other companies, other video games, with different characteristics.

7. Conclusion

In this paper, we present a comprehensive multi-case study on reuse in the field of game software engineering. The study combines multiple sources of information, including questionnaires, interviews, access to source code, and the use of phylogenetic trees. Our work confirms the claims of previous works about the widespread use of game engines, the motivations that developers may have for reuse, and the charactristics of a development team for reuse [12, 13, 14]. Furthermore, our study reveals the role of prefabs as fundamental units for facilitating the reuse and management of objects in game development. Prefabs occupy their own reuse space compared to classes or the prototype pattern. Our results show how prefabs evolve and how developers can leverage this evolution to gain insight into reuse. However, our results also highlight potential risks, including the introduction of redundancy, bugs, and unused prefabs, which can lead to prefab hell. Moreover, prefabs can also play a role in reuse among video games when it comes to developing a sequel.

Despite the relevance of prefabs for reuse, prefabs have been under-the-radar for previous research efforts [2, 3]. For the first time in scientific literature, our work sheds light on prefabs. This has been achieved by means of a multiple case study powered by data source triangulation, observer triangulation, and methodological triangulation. Deepening the understanding of prefabs may inspire new ideas to improve reuse practices not only in game software engineering but also in software engineering in general. Future work should involve conducting more case studies to validate and expand upon the findings presented in this paper. Also, the application of phylogenetic trees could be explored to analyze how prefab relationships evolve over time. Examining these temporal dynamics may reveal patterns of adaptation and reuse that remain hidden in a static analysis, offering new insights into the evolution of game assets.

8. Replication Package

The replication package of this work can be found at `https://doi.org/10.5281/zenodo.14591667`

9. Acknowledgements

This work was supported in part by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the Project VARIATIVA under Grant PID2021-128695OB-I00, and in part by the Gobierno de Aragón (Spain) (Research Group T61_23R). This work was partially supported by the Spanish Ministry of Science, Innovation and Universities under the Project VARNETICA (CNS2023-145422).

We would like to express our gratitude to the companies and developers who have provided invaluable assistance for the realization of this work.

References

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

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

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

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

[5] X. Chen, M. Usman, D. Badampudi, Understanding and evaluating software reuse costs and benefits from industrial cases—a systematic literature review, t (2024) 107451.

[6] C. W. Krueger, Software reuse, ACM Computing Surveys (CSUR) 24 (2) (1992) 131–183.

[7] C. Wohlin, Case study research in software engineering—it is a case, and it is a study, but is it a case study?, Information and Software Technology 133 (2021) 106514. doi:https://doi.org/10.1016/j.infsof.2021.106514.
URL https://www.sciencedirect.com/science/article/pii/S0950584921000033

[8] C. Wohlin, A. Rainer, Is it a case study?—a critical analysis and guidance, Journal of Systems and Software 192 (2022) 111395. doi:https://doi.org/10.1016/j.jss.2022.111395.
URL https://www.sciencedirect.com/science/article/pii/S0164121222001121

[9] R. K. Yin, Case study research: Design and methods, Vol. 5, sage, 2009.

[10] P. Runeson, M. Host, A. Rainer, B. Regnell, Case study research in software engineering: Guidelines and examples, John Wiley & Sons, 2012.

[11] S. Navarro, A. Iglesias, P. Francisca, C. Cetina, J. Font, Phylogenix: Bringing phylogenetics to unity, in: 28th ACM International Systems and Software Product Line Conference, Vol. A, 2024, pp. 38–41.

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

[13] B. J. Geisler, S. L. Kavage, A multi-engine aspect-oriented language with modeling integration for video game design, in: Evaluation of Novel Approaches to Software Engineering: 15th International Conference, ENASE 2020, Prague, Czech Republic, May 5–6, 2020, Revised Selected Papers 15, Springer, 2021, pp. 336–359.

[14] S. Aleem, L. F. Capretz, F. Ahmed, A digital game maturity model (dgmm), Entertainment Computing 17 (2016) 55–73.

[15] A. Ampatzoglou, A. Chatzigeorgiou, Evaluation of object-oriented design patterns in game development, Information and Software Technology 49 (5) (2007) 445–454.

[16] J. Kessing, T. Tutenel, R. Bidarra, Designing semantic game worlds, in: Proceedings of the The third workshop on Procedural Content Generation in Games, 2012, pp. 1–9.

[17] T. Olsson, D. Toll, A. Wingkvist, M. Ericsson, Evaluation of a static architectural conformance checking method in a line of computer games, in: Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures, 2014, pp. 113–118.

[18] T. Ollsson, D. Toll, A. Wingkvist, M. Ericsson, Evolution and evaluation of the model-view-controller architecture in games, in: 2015 IEEE/ACM 4th International Workshop on Games and Software Engineering, IEEE, 2015, pp. 8–14.

[19] A. I. Wang, N. Nordmark, Software architectures and the creative processes in game development, in: Entertainment Computing-ICEC 2015: 14th International Conference, ICEC 2015, Trondheim, Norway, September 29-Ocotober 2, 2015, Proceedings 14, Springer, 2015, pp. 272–285.

[20] M. Ghoreshi, H. Haghighi, An incremental method for extracting tests from object-oriented specification, Information and Software Technology 78 (2016) 1–26.

[21] F. M. Boaventura, V. T. Sarinho, Mendiga: A minimal engine for digital games, International Journal of Computer Games Technology 2017 (1) (2017) 9626710.

[22] F. Nunnari, A. Heloir, Write-once, transpile-everywhere: re-using motion controllers of virtual humans across multiple game engines, in: Augmented Reality, Virtual Reality, and Computer Graphics: 5th International Conference, AVR 2018, Otranto, Italy, June 24–27, 2018, Proceedings, Part I 5, Springer, 2018, pp. 435–446.

[23] V. Khanve, Are existing code smells relevant in web games? an empirical study, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 1241–1243.

[24] W. K. Mizutani, F. Kon, Unlimited rulebook: A reference architecture for economy mechanics in digital games, in: 2020 IEEE International Conference on Software Architecture (ICSA), IEEE, 2020, pp. 58–68.

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

[26] J. Chueca, J. I. Trasobares, Á. Domingo, L. Arcega, C. Cetina, J. Font, Comparing software product lines and clone and own for game software engineering under two paradigms: Model-driven development and code-driven development, Journal of Systems and Software 205 (2023) 111824.

[27] V. Nardone, B. Muse, M. Abidi, F. Khomh, M. Di Penta, Video game bad smells: What they are and how developers perceive them, ACM Transactions on Software Engineering and Methodology 32 (4) (2023) 1–35.

[28] V. R. Basili, H. D. Rombach, The tame project: Towards improvement-oriented software environments, IEEE Transactions on software engineering 14 (6) (1988) 758–773.

[29] G. Weyenberg, R. Yoshida, Reconstructing the phylogeny: Computational methods, in: Algebraic and Discrete Mathematical methods for modern Biology, Elsevier, 2015, pp. 293–319.

[30] M. Nei, Molecular evolutionary genetics, Columbia university press, 1987.

[31] J. Chueca, D. Blasco, C. Cetina, J. Font, Leveraging phylogenetics in software product families: The case of latent content generation in video games, in: 28th ACM International Systems and Software Product Line Conference, Vol. A, 2024, pp. 113–124.

[32] C. Darwin, On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life, John Murray, 1859.

[33] D. A. Baum, S. D. Smith, Tree thinking: an introduction to phylogenetic biology, in: Tree thinking: An introduction to phylogenetic biology, 2012, pp. 476–476.

[34] G. A. Pavlopoulos, T. G. Soldatos, A. Barbosa-Silva, R. Schneider, A reference guide for tree analysis and visualization, BioData mining 3 (2010) 1–24.

[35] Liquid Web, Video game industry: Statistics and trends (2024) —, https://www.liquidweb.com/blog/video-game-statistics/, [Online; accessed 17-March-2024] (2024).

[36] KONVOY, The era of the indie game, https://www.konvoy.vc/content/the-era-of-the-indie-game, [Online; accessed 17-March-2024] (2024).

[37] ROCKETBRUSH, Top video game genres in 2024: Revenue, statistics, https://rocketbrush.com/blog/most-popular-video-game-genres-in-2024-revenue-\statistics-genres-overview, [Online; accessed 17-March-2024] (2024).

[38] Game Developers Conference, State of the game industry

2024, https://images.reg.techweb.com/Web/UBMTechweb/%7B4fe03be1-d6b1-4f91-95f4-b4bdea9e739e%7D_GDC24-SOTI-Report_Final.pdf, [Online; accessed 22-July-2024] (2024).

[39] Unreal, Blueprint class — unreal engine 4.27 documentation, [Online; Accessed on 10-July-2024] (2014).
URL https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/ClassBlueprint/

[40] Godot Communtiy, Nodes and scenes - godot docs, [Online; Accessed on 10-July-2024] (2014).
URL https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html

[41] I. Letunic, P. Bork, Interactive tree of life (itol) v6: recent updates to the phylogenetic tree display and annotation tool, Nucleic Acids Research (2024) gkae268.