



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Understanding and Tackling the Challenges of Game Software Engineering

FEBRUARY 2026

Author:

Javier Verón Mérida

Thesis Supervisors: Dr. Óscar Pastor López

Dra. M^a Francisca Pérez Pérez

Dedication

To my family, for always supporting and helping me in every possible way.

To my friends, for always being present when we needed each other.

Acknowledgments

First, I want to thank my directors, Dr. Óscar Pastor and Dr. Francisca Pérez, for guiding me and helping me every step of the way. Thank you for putting your time and trust in me. I also want to mention Dr. Carlos Cetina, as he has been a light through my studies, constantly showing roads that I could not have seen otherwise.

I want to thank my colleagues at the SVIT Research Group, at Universidad San Jorge, and at Entalto Studios. Every one of you is the best example of how a real team member should be. I especially want to mention Jorge Chueca for having traveled so many paths with me since 2016.

Finally, thank you to my loved ones for always supporting me and understanding that I could not be around as much as I would have wanted to during these past years.

Abstract

Video game development differs in certain aspects from traditional software development. Therefore, although it can benefit from much of the research focused on traditional Software Engineering, it also requires work that addresses the specific challenges of the video game industry. However, research in Game Software Engineering (GSE) is a field with significantly fewer studies and still-evolving direction and standards.

To delve deeper into the current challenges of GSE, a Systematic Literature Review (SLR) was conducted as part of this work. The SLR revealed an open and highly demanded challenge: Requirements Traceability (TLR). Additionally, it identified a lack of research on green computing. Consequently, the objective of this thesis is to present: an approach for TLR in GSE with an industrial case study; and a comparative analysis focused on energy consumption between two different implementation approaches in the Unreal Engine 5 game engine.

The proposed approach for TLR in GSE outperforms the reference model in all performance metrics. The comparative analysis shows that certain operations consume significantly more energy when implemented using the engine's visual scripting language than when using C++ code. Furthermore, the study reveals that measuring the energy consumption of video games as if they were traditional software leads to overlooking key

aspects. Therefore, this research proposes an additional metric: frames per second.

This thesis presents a compendium including three papers published in academic journals and conferences, developed during the research process: An SLR showing a steady increase in GSE research since 2009; the use of Non-Player Character (NPC) simulations reduces the search space by 99% and significantly improves TLR in real-world commercial GSE cases; and the energy impact of using a modeling language versus C++ code. Given the widespread popularity of video games, this work reveals that the energy cost could reach the equivalent of 28 million European households.

Resumen

El desarrollo de videojuegos difiere en ciertos aspectos del desarrollo de software tradicional y, por tanto, aunque se pueda beneficiar de buena parte de la investigación enfocada en Ingeniería del Software tradicional, necesita trabajos que traten los problemas particulares de la industria de los videojuegos. Sin embargo, la investigación en Ingeniería del Software enfocada en Videojuegos, GSE (Game Software Engineering), es un campo con notablemente menos trabajos de investigación y con una dirección y estándares todavía en desarrollo.

Para profundizar en los retos actuales de GSE, durante la realización de este trabajo se realiza una Revisión Sistemática de la Literatura (SLR). La SLR ha permitido detectar un primer reto de TLR (trazabilidad de requisitos) que todavía está abierto, además de ser un problema muy demandado. Además, la SLR ha permitido detectar una falta de trabajos sobre computación verde. En consecuencia, el objetivo de esta tesis es presentar: una aproximación para TLR en GSE con un estudio de caso industrial; y un análisis comparativo centrado en el consumo energético entre dos aproximaciones diferentes de implementación en el motor de videojuegos Unreal Engine 5.

La aproximación propuesta para TLR en GSE supera a la de referencia en todas las medidas de rendimiento. El análisis comparativo realizado muestra que ciertas operaciones consumen notablemente más cantidad

de energía al implementarse con el language de modelado ofrecido por el motor que al implementarse con código C++. También, el análisis realizado revela que medir el consumo energético de videojuegos como el de software tradicional provoca que se pierdan aspectos importantes, por lo que esta investigación propone otra medida que tener en cuenta: los fotogramas por segundo.

Esta tesis presenta un compendio que incluye tres artículos publicados en revistas y conferencias académicas realizados durante el proceso de investigación: Una SLR con resultados que muestran un aumento constante en la investigación de GSE desde 2009; la utilización de simulaciones de Non-Player Characters (NPCs) reduce el espacio de búsqueda en un 99% y mejora significativamente TLR en casos comerciales reales en GSE; y el impacto energético en el desarrollo utilizando un lenguaje de modelado respecto a código C++. Dada la gran popularidad de los videojuegos, este trabajo revela que el coste en energía podría alcanzar el equivalente a 28 millones de hogares europeos.

Resum

El desenvolupament de videojocs difereix en certs aspectes del desenvolupament de programari tradicional i, per tant, encara que es puga beneficiar de bona part de la investigació centrada en Enginyeria del Programari tradicional, necessita treballs que tracten els problemes particulars de la indústria dels videojocs. No obstant això, la investigació en Enginyeria del Programari aplicada als Videojocs (GSE, per les seues sigles en anglés) és un camp amb notablement menys estudis i amb una direcció i uns estàndards encara en desenvolupament.

Per a aprofundir en els reptes actuals de la GSE, durant la realització d'aquest treball s'ha dut a terme una Revisió Sistemàtica de la Literatura (SLR). Aquesta SLR ha permés detectar un primer repte de TLR (traçabilitat de requisits) que continua obert, a més de ser molt sol·licitat. També ha posat de manifest la manca de treballs sobre computació verda. En conseqüència, l'objectiu d'aquesta tesi és presentar: una aproximació per a la TLR en GSE amb un estudi de cas industrial; i una anàlisi comparativa centrada en el consum energètic entre dues aproximacions diferents d'implementació al motor de videojocs Unreal Engine 5.

L'aproximació proposada per a la TLR en GSE supera la de referència en totes les mesures de rendiment. L'anàlisi comparativa realitzada mostra que certes operacions consumeixen notablement més energia quan s'implementen amb el llenguatge de modelatge que ofereix el motor que

amb codi en C++. A més, l'anàlisi revela que mesurar el consum energètic dels videojocs com si foren programari tradicional fa que es perdin aspectes importants, i per això aquesta investigació proposa tindre en compte una altra mesura: els fotogrames per segon.

Aquesta tesi presenta un compendi que inclou tres articles publicats en revistes i congressos acadèmics durant el procés d'investigació: Una SLR amb resultats que mostren un augment constant en la investigació sobre GSE des de 2009; l'ús de simulacions de Personatges No Jugadors (NPCs) redueix l'espai de cerca en un 99% i millora significativament la TLR en casos comercials reals de GSE; i l'impacte energètic d'utilitzar un llenguatge de modelatge respecte al codi en C++. Donada la gran popularitat dels videojocs, aquest treball revela que el cost energètic podria arribar a l'equivalent de 28 milions de llars europees.

Contents

Dedication	iii
Acknowledgments	v
Abstract - Resumen - Resum	vii
Contents	xiii
I Introduction	1
II Compendium of Articles	19
1 The Consolidation of Game Software Engineering: A Systematic Literature Review of Software Engineering for Industry-scale Computer Games	21

2 Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering	95
3 A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD	121
III Discussion	157
IV Conclusions	173

Part I

Introduction

Introduction

This part introduces the motivation, context, and objectives of this thesis. It includes an overview of the research process and describes the methodology applied. The last sections of the chapter present the structure of this work and the scientific articles that are included in this document.

Motivation and context

This thesis focuses on understanding and tackling the challenges of Game Software Engineering in relation to traditional Software Engineering and addresses new ways of leveraging specific domain assets of video games in Software Engineering research.

Game Software Engineering

The video game industry has experienced exponential growth in recent decades. This unprecedented growth has led to greater complexity in video game development, requiring specialized software engineering practices to ensure the quality, stability, and efficient production of these complex software systems.

While video games share some similarities with traditional software, their unique nature merging artistic creativity with technical precision presents specific challenges that require a differentiated approach. Video games incorporate various disciplines, including programming, game design, art, music, animation, and storytelling. This convergence of fields creates the immersive digital worlds that we all can enjoy in video games, but also presents specific domain challenges at the same time.

Game Software Engineering (GSE) emerges as an independent discipline within software engineering, focusing on the unique challenges posed by video game development. The need to create products that are 'fun' and meet the expectations of a constantly evolving industry has led to significant differences between GSE and traditional software engineering. The differences between GSE and traditional software engineering can be seen in several aspects, e.g., the widespread usage of game engines, heterogeneous teams provoking the popularization of software models, or non-player characters (NPCs), e.g., adversaries in a racing game, being a very common feature.

Objectives and Research Process

Research in GSE is crucial to address emerging challenges in this rapidly evolving field. The proliferation of publishing platforms (such as PlayStation 5 [1], Nintendo Switch [2], and Xbox Series X and S [3]), the increasing complexity of games, and the demand for innovative user experiences require the development of new techniques and specific tools specific for this kind of software. However, research in GSE is still a growing field where topics such as Traceability Links Recovery (TLR), Green Computing, or Software Product Lines (SPL) are still in their early stages and have a lot of room for improvement; while Software Engineering is a more mature field that serves as foundation for GSE. With this landscape as the current state of research in GSE, the Research Questions that this thesis aims to answer are the following:

RQ₁: *How have Software Engineering practices for industry-scale computer games evolved, and what are the key challenges and future research directions identified in recent literature?*

RQ₂: *Can traditional Software Engineering research approaches be leveraged for the key challenges of Game Software Engineering?*

Research Development Progress

Figure 1 shows an overview of this thesis work, where video game development peculiarities (left part of the figure) and Software Engineering research (right part of the figure) meet in GSE (central part of the figure). The central part of Figure 1 also shows the research works that are published during this thesis, which are introduced next. IST 165 [4] shows a study on the current state of the art in GSE research, where industry-scale video games inherent characteristics are taken into account when researching on how to perform or adapt software engineering practices and research topics. The results of IST 165 allowed us to address RQ₁ and to delve into the differences between traditional Software Engineering and GSE, bringing us to a first challenge in TLR, which is still open. In addition, a gap in green computing was detected. To address this, CAISE 24 [5] and MODELS 24 [6] are the resulting publications during

this thesis. Both of these last works allowed us to address RQ_2 , from the results of IST 165 in RQ_1 .

Next, the publications of this thesis are summarized.

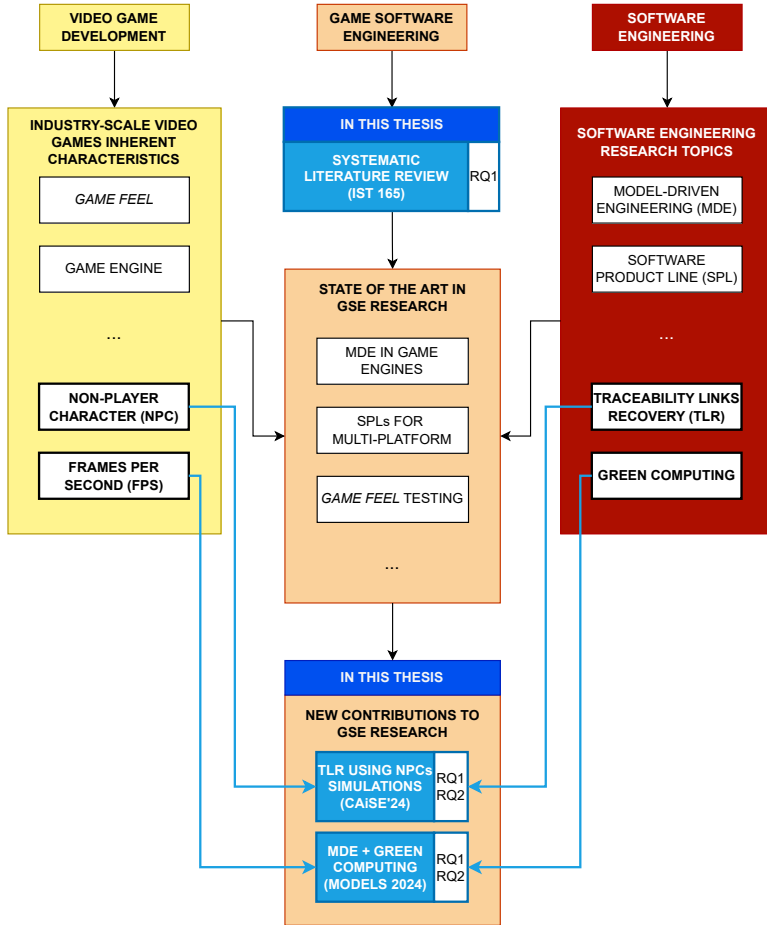


Figure 1: Thesis work overview

Systematic Literature Review

The first research work included in Figure 1, IST 165 [4], is a Systematic Literature Review (SLR) on industry-scale GSE research.

A Systematic Literature Review (SLR) is a research method that comprehensively collects and analyzes existing literature on a specific topic. It follows a rigorous and transparent process to minimize bias and ensure the quality of results. In addition, SLRs typically evaluate and synthesize findings, identifying patterns, gaps, and trends in current knowledge to inform future research. This makes SLRs valuable for providing a global view of the state of knowledge, highlighting solid evidence and areas of uncertainty, and guiding best practices. By offering a structured analysis, SLRs help practitioners and researchers make informed decisions and direct future efforts effectively.

The first paper included in this thesis, IST 165 [4], provides a comprehensive analysis of research in GSE, shedding light on the current state of the field and identifying areas where further research is needed. After analyzing 98 industrial-scale GSE studies, this SLR reveals significant growth in research activity and maturity in the field since 2009.

The findings of this SLR support the idea that GSE is a growing research field presenting new forms of software creation. The SLR highlights the need for greater collaboration between academia and industry to bridge the gap between research and practice. By engaging industry professionals, academia can gain valuable insights into real-world challenges, leading to the development of more effective solutions.

The SLR contributes significantly to the GSE knowledge base by providing a comprehensive overview of existing research and identifying future directions. As the video game industry continues to expand, GSE research is proved essential to drive innovation and ensure the development of high-quality games that meet the demands of a constantly changing global audience.

This SLR was developed following the guidelines given by Kitchenham and Charters [7], which are widely accepted in the community.

The results of the SLR lead to the identification of several potential future research directions for GSE and prove that GSE has enough volume and interest by the community to be considered a whole domain instead of a small software engineering subtopic. Particularly, there are still open challenges in TLR for GSE and a gap in green GSE.

Empirical Approaches

Traceability Link Recovery

CAISE 24 [5] is a research work that is made during this thesis to address the challenge identified in the SLR regarding Traceability Link Recovery (TLR).

TLR is a critical task in software engineering. It involves the identification of relationships and dependencies among various software artifacts such as requirements, design, source code, and tests. TLR is essential for tasks like software maintenance, requirement management, impact analysis, and quality improvement. It helps developers to understand the implications of changes, ensures requirements are correctly implemented, identifies affected areas of the system, and detects inconsistencies, thereby enhancing software quality.

In GSE, TLR faces unique challenges due to the complexity of modern games, including large codebases, integration of art and software, and rapid iterative development. The second paper that is included in this thesis, CAISE 24 [5], explores the use of video game simulations to enhance TLR in GSE. Simulations can be found in non-player characters (NPCs), which are a common element in video game development. NPCs are entities controlled by the CPU for populating the game world and even guiding, teaming, or opposing the player. In this work, it is argued that they can be leveraged to provide valuable information for GSE tasks; in this case, reducing the search space of the video game source code to find connections between game requirements and the source code. NPCs are used to reproduce specific game scenarios related to the requirements being traced. During the simulation, the executed code is logged, cre-

ating a filtered code subset that represents the simulated scenario and contains code relevant to the requirement.

Both the requirements and the filtered code are processed using NLP techniques to unify the language of the software artifacts and facilitate the TLR process. Techniques include syntactic analysis, stemming, and human NLP. Finally, an evolutionary algorithm generates, evaluates, and evolves a population of code fragments, converging on a ranking of possible solutions for the requirement. The algorithm uses Topic Modeling as a fitness function, evaluating the relevance of each code fragment in relation to the processed requirement.

The evaluation of this work was performed on a commercial video game that was released in PC and PlayStation 4. The results of this evaluation show that the inclusion of simulations significantly reduces the search space for TLR by an average of 99.21%. This allows the evolutionary algorithm to function more efficiently and accurately. The simulation-based approach significantly outperforms a state-of-the-art technique that does not use simulations, with improvements of 23.23% in recall, 18.34% in precision, and 22.87% in the F-measure.

Finally, a focus group of game software engineers confirmed a preference for the simulation-based approach, highlighting its usefulness for finding requirements in different game genres and more complex problems.

The proposed approach that leverages NPCs simulations is relevant beyond the scope of this work, as this approach could be applied to other maintenance tasks, such as bug localization, and that future research may explore its application in different video game development contexts.

Green Computing

MODELS 24 [6] is a research work as a result of this thesis on Green Computing for video games.

Green Computing, also known as sustainable computing, involves designing, developing, and using information and communication technologies (ICT) in energy-and-resource-efficient ways. Its goal is to minimize the

environmental impact of ICT throughout its life cycle, from production and use to disposal.

In the gaming industry, where creating complex 3D worlds demands high performance and significant energy consumption, adopting Green Computing practices can reduce energy usage, thus extending battery life on mobile devices, and lowering the carbon footprint of the sector. Given the vast number of gamers worldwide, even small improvements in energy efficiency can have a substantial positive impact on environmental sustainability.

The third paper included in this thesis, MODELS 24 [6], compares and analyzes the energy consumption of visual scripting models and C++ code in Unreal Engine 5 [8] (UE5), with the aim of raising awareness about the importance of applying Green software development and Green model-driven development (Green MDD) knowledge and techniques to video game development.

In this work, three relevant video game components were implemented twice: once using visual scripting models, which are called Blueprints in UE5; and once using C++ code. These components are: health system management (tracks and updates the game character's health points based on damage received), input processing (captures and interprets player inputs; e.g., keyboard, mouse, controller; to trigger in-game actions such as movement through the game world), and inventory search operations (allows efficient lookup, filtering, or sorting of items in a player's items inventory). The framework that we used for the measurements, comparison, and analysis is the FEETINGS framework, which has been validated by the green computing community. Power consumption was measured for the hard drive, graphics card, processor, and the entire device under test (DUT).

In Green Computing works on energy consumption measurement, measures take into account the total execution time for analyzing consumption. However, in video games, the time of the execution is not as important as the time required for rendering a frame, as games can run at very different frame rates, which ends up in different consumption amounts for the same total execution time. This is why the energy consumption

of each component in both implementations was compared taking into account the consumption per frame.

It was observed that using Blueprint models can result in higher energy consumption per frame compared to C++ code for some components. Specially, the Blueprints implementation of the inventory operations component showed a significant increase in consumption per frame, compared to the C++ implementation.

This work seeks to raise awareness about the importance of green software development in the context of video games and encourages the industry to consider the environmental impact of their design and implementation decisions: although Blueprint models may be convenient for development, it is crucial to consider their potential impact on energy consumption, especially in games that require high performance. The study demonstrates that the choice of implementation approach (Blueprint models or C++) can significantly influence the energy consumption and performance of video games.

The need for future research is highlighted to explore the energy consumption of other common video game components and across different target platforms, to better understand the implications of Green MDD in the video game industry.

Research Methodology

This thesis was planned based on the methodology proposed by Roel J. Wieringa in *Design Science Methodology for Information Systems and Software Engineering* [9]. The main motivation behind the Design Science methodology is the improvement of practice through the design and research of artifacts in context. The goal is to solve real-world problems and generate useful knowledge for stakeholders. It is an iterative approach that combines solution design with empirical research to assess its effectiveness. The Empirical Cycle used in this methodology is a rational process for answering scientific knowledge questions.

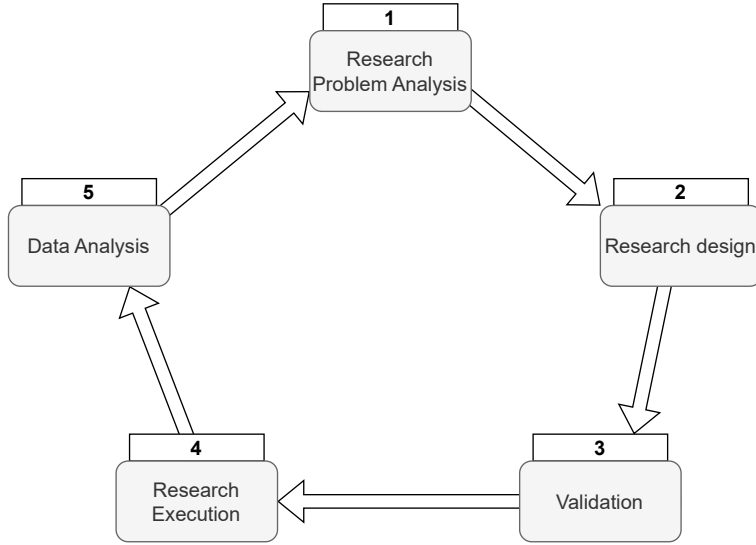


Figure 2: Empirical cycle from the Design Science methodology.

The articles included in the thesis used the Design Science methodology, applying the empirical cycle as shown in Figure 2, consisting of the following phases:

1. **Research Problem Analysis:** This is the starting point where the research problem to be solved is clearly defined. It involves framing the problem and formulating the knowledge questions that will guide the research.
2. **Research design:** In this phase, the research setup is designed, including the selection or construction of the objects of study, the definition of the variables to be measured and the measurement instruments, and the design of the inferences to be made from the data.
3. **Validation:** Validation in the empirical cycle seeks to ensure that the research setup is appropriate for answering the knowledge ques-

tions. For the validation, is important to take into consideration if the research setup allows the planned inferences to be made, if other researchers can replicate the research, and if ethical standards are respected in the research, when applicable. All the included studies incorporate measurements commonly accepted by the research community and extensively used in the relevant literature and made sure to offer everything needed for replication purposes. In addition, we studied the threats to the validity of our approaches. The impact of these threats has been mitigated or minimized whenever possible from the design of the experiments.

4. **Research Execution:** The research is carried out according to the established design, collecting data from the objects of study.
5. **Data Analysis:** The collected data is analyzed to answer the knowledge questions formulated at the beginning of the cycle.

Structure of the Thesis

This work is structured in accordance to the regulations in Universitat Politècnica de València with regard to PhD thesis development presented by a compendium of articles. It is organized in the following four parts:

- I. **Introduction:** The first part of the thesis is a description of the motives and context in which the research work takes place, the objectives of this work and its process, the methodology applied through the research, and a brief summary of the articles that comprise the compendium.
- II. **Compendium of Articles:** The second part includes the collection of articles included in the compendium. All of these articles were published in specialized journals or presented in specialized conferences. They keep the original content and bibliographies, although they are presented following the format for the thesis in accordance to the regulations in Universitat Politècnica de València.

-
- III. Discussion:** The third part presents a discussion of the results and contributions obtained through the research and the future works that arise.
- IV. Conclusions:** The fourth and last part of the thesis presents the conclusions of this work from the perspective of the objectives that were identified and the results that are obtained throughout this work.

Compendium Description

The articles listed below were developed and published as part of the research process presented in this thesis:

- 1. The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games** (Information and Software Technology) (Chueca, Verón, Font, Pérez, Cetina, 2023) (**IST 165**) [4].
This work is a Systematic Literature Review (SLR) on Game Software Engineering (GSE) at an industrial scale. The article examines trends in GSE from 2009 to 2021, identifying a significant increase in research in the field, as well as an evolution toward more complex topics such as architecture and design. The study also identifies future research areas and highlights the growing convergence between academic research and industrial practice in this field.
- 2. Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering** (International Conference on Advanced Information Systems Engineering) (Verón, Lapeña, Cetina, Pastor, Pérez, 2024) (**CAISE 24**) [5].
This work analyzes the impact of using simulations based on Non-Player Characters (NPCs) on Traceability Link Recovery (TLR) in video game development. The article explores how these simulations can reduce the search space and improve TLR accuracy. The effectiveness of the proposed approach is evaluated in a commercial

video game, Kromaia, showing promising results and developers' preference for this approach.

3. A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD (Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems) (Verón, Pérez, Calero, Moraga, Pérez, Cetina, 2024) (**MODELS 24**) [6].

This work explores the energy consumption of Visual Scripting models in Unreal Engine 5; named as "Blueprints"; compared to C++ code. The energy consumption of three common game components is analyzed: health system management, input processing, and inventory operations. The FEETINGS framework is used to measure and analyze energy consumption, showing that C++ implementation is more energy-efficient than Blueprint models for two of the three components, with the difference for inventory operations being quite large. This work underscores the need for greater awareness of energy consumption in the video game industry and highlights the importance of sustainability-driven modeling tools and documentation on the topic.

Additionally, all of the articles included in the compendium were presented in relevant national and international conferences as follows:

- **IST 165** was presented as a journal-first paper in the 2nd Spanish Conference on Video Games (CEV 2023).
- **CAISE 24** was presented in the 36th International Conference on Advanced Information Systems Engineering (CAISE 24).
- **MODELS 24** was presented in the 27th International Conference on Model Driven Engineering Languages and Systems (MODELS) and it was additionally presented in the Devcom Developer Conference 2024 (DDC 2024). DDC is the biggest international industrial conference on video game development in Europe, which made that presentation a great opportunity to bring closer industry and academia.

Research Support

The research works included as a compendium in this thesis have been partially funded as follows:

- **IST 165** [4] 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 S05_20D).
- **CAISE 24** [5] was partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by Gobierno de Aragón (Research Group S05_20D), and by Centro para el Desarrollo Tecnológico y la Innovación (CDTI) (Reference: SAV-20221011).
- **MODELS 24** [6] was partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by UE under the Project OASSIS (PID2021-122554OBC31/AEI/10.13039/501100011033/FEDER), by CECD (JCCM) and FEDER funds under the Project EMMA (SBPLY/21/180501/000115), by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/ under the project SEEAT (PDC2022-133249-C31), and by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/PRTR under the project PLAGEMIS (TED2021-129245B-C22).

Bibliography

- [1] Sony Interactive Entertainment Europe Limited (SIEE). *PlayStation®5*. [Online; accessed 25-November-2024]. 2024. URL: <https://www.playstation.com/es-es/ps5/> (cit. on p. 6).
- [2] Ltd. Nintendo Company. *Nintendo Switch*. [Online; accessed 25-November-2024]. 2024. URL: <https://www.nintendo.com/es-es/Hardware/Familia-Nintendo-Switch/Nintendo-Switch/Nintendo-Switch-1148779.html> (cit. on p. 6).

-
- [3] Microsoft Corporation. *Xbox Consoles*. [Online; accessed 25-November-2024]. 2024. URL: <https://www.xbox.com/es-ES/consoles> (cit. on p. 6).
 - [4] Jorge Chueca et al. “The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”. In: *Information and Software Technology* (2023) (cit. on pp. 6, 8, 15, 17).
 - [5] Javier Verón et al. “Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2024, pp. 352–367 (cit. on pp. 6, 9, 15, 17).
 - [6] Javier Verón et al. “A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 2024, pp. 114–125 (cit. on pp. 6, 10, 11, 16, 17).
 - [7] Barbara Kitchenham and Stuart Charters. “Guidelines for performing systematic literature reviews in software engineering”. In: (2007) (cit. on p. 8).
 - [8] Epic Games Inc. *The most powerful real-time 3D creation tool - Unreal Engine*. <https://www.unrealengine.com/>. [Online; accessed 26-March-2024]. 2024 (cit. on p. 11).
 - [9] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014 (cit. on p. 12).

Part II

Compendium of Articles

Chapter 1

The Consolidation of Game Software Engineering: A Systematic Literature Review of Software Engineering for Industry-scale Computer Games

Context: Game Software Engineering (GSE) is a branch of Software Engineering (SE) that focuses on the development of video game applications. In past years, GSE has achieved enough volume, differences from traditional software engineering, and interest by the community to be considered an independent scientific domain, veering out from traditional SE. **Objective:** This study evaluates the current state of the art in software engineering for industry-scale computer games identifying gaps and consolidating the magnitude and growth of this field. **Method:** A Systematic Literature Review is performed following best practices to ensure the relevance of the studies included in the review. We analyzed 98 GSE studies to extract the current intensity, topics, methods, and quality of GSE. **Results:** The GSE research community has been growing over the years, producing over four times more research than before the previous GSE survey. However, this community is still very dispersed, with no main venues holding most of the GSE scientific studies. A broader range of topics is covered in this area, evolving towards those of a mature field such as architecture and design. Also, the reviewed studies employ more elaborated empirical research methods, even though the study reports need to be more rigorous in sections related to the critical examination of the work. **Conclusion:** The results of the SLR lead to the identification of 13 potential future research directions for this domain. GSE is an independent, mature, and growing field that presents new ways of software creation where the gap between industry and academia is narrowing. Video games present themselves as powerful tools to push the boundaries of software knowledge.

Author's version of the article: J. Chueca, J. Verón, J. Font, F. Pérez, and C. Cetina, "The Consolidation of Game Software Engineering: A Systematic Literature Review of Software Engineering for Industry-scale Computer Games", in *Information and Software Technology*, 2023, vol. 165, doi: 10.1016/j.infsof.2023.107330.

1.1 Introduction

In 2021, the video game industry generated total revenues of \$180.3 billion (up 1.4% over 2020 [1]), and it keeps growing after having already surpassed the revenues of the movie and music industries combined in 2019 [2], this makes it the largest entertainment industry in terms of revenue.

Video games are advanced software products and, at the same time, they are complex works of creativity and art [3]. This merge of disciplines, the need for the product to be fun, and the unrelenting growth of the video game industry have created significant differences between traditional Software Engineering (SE) and Game Software Engineering (GSE) [4]. These conditions require constant and fast communication among developers, flexible design, and maintainable implementation. Moreover, video games tend to be products with a much more limited life cycle than other software products [5]. In Section 1.2, we present more differences and similarities between GSE and traditional SE, showing that the differences outnumber the similarities. For these reasons, we propose that GSE be considered an independent domain that presents divergence from traditional software engineering.

1.1.1 Goals and Contributions

Many authors [6, 7, 4, 5] argue that GSE and traditional SE differ more than they might be currently conceived. We aim to show whether or not GSE has enough volume and interest by the community to be considered a whole domain instead of a small software engineering subtopic.

In addition, we also want to shed light on the current state of research on software engineering techniques, approaches, and methods being applied to computer games and to compare their evolution before and after 2009 (last year covered by the previous GSE survey [5]), in order to provide background information for any relevant GSE work in the future. This background information will also benefit both experts and newcomers to this field by showing the tendencies and gaps in current research.

Furthermore, the results of the survey lead to the identification of several potential future research directions for GSE.

Finally, this study tightens the gap between industry and academia, showing both of them how to nourish each other. We present a large set of industry-scale research that invites practitioners to publish their expertise in research venues, and we show SE researchers that GSE is a fertile domain with plenty of case studies to build up the knowledge of software.

1.1.2 Lack of Existing SLRs

Some authors have reviewed different topics of video games as a software domain. However, there is a dearth of systematic reviews on more general topics such as Software Engineering. The three main approaches to reviewing the topic of video games as software have been **surveys and interviews** (six studies), **grey literature** (seven studies), and **academic literature** (nine studies). Table 1.1 shows the studies that we found about GSE, followed by a brief commentary of each study.

The **surveys and interviews** that deal with the domain of games as software are the following: Murphy et al. [8] surveyed 14 game and non-game developers, finding different practices among these domains. Kasurinen et al. [9] conducted 27 interviews with professional game developers to understand how they test their products. Nandhakumar et al. [10] surveyed and observed three different development companies to explore the challenges of conceptualizing and realizing the desired user experience. Kasurinen et al. [6] interviewed 33 professionals from different game development companies, focusing on the particular characteristics of video game business topics. McKenzie et al. [11] interviewed 12 participants from New Zealand’s game industry, revealing that custom agile methodologies are the leading practice for developers. Musil et al. [12] created a questionnaire for Austrian game studios relating scope and industry-specific problems.

Some contributions are only reported in the **grey literature** and at professional venues such as the Game Developers Conference. Politowski et al. have surveyed postmortem project analyzes from the grey literature

Table 1.1: Scientific reports on GSE.

Approach	Studies	Total
Surveys and Interviews	Murphy et al. [8] Kasurinen et al. [9] Nandhakumar et al. [10] Kasurinen et al. [6] McKenzie et al. [11] Musil et al. [12]	6
Grey Literature Reviews	Politowski et al. [13] Politowski et al. [14] Politowski et al. [15] Politowski et al. [16] Ullmann et al. [17] Petrillo et al. [18] Washburn et al. [19]	7
Academic Literature Reviews	Engström et al. [3] Mizutani et al. [20] Viana et al. [21] Zhu et al. [22] Politowski et al. [23] Osborne et al. [24] Vargas et al. [25] Aleem et al. [26] Ampatzoglou et al. [5]	9

to find what software engineering processes are being used in the video game industry [13] and the main development problems in the current industry [14]. They have also surveyed grey literature on existing testing processes [15]. Politowsky et al. created a dataset of more than 200 postmortems gathering 1,035 problems related to software engineering in video games [16]. Ullmann et al. read 440 postmortem problems to identify anti-patterns in game development [17]. Petrillo et al. [18] examined 20 postmortems to find how agile methodologies are applied in video game development and how the lack of a rigorous methodology is related to an increase in problems found in development. Washburn et

al. [19] analyzed 155 postmortems, finding best practices and pitfalls in game development.

Other researchers have sought answers by reviewing **academic literature**. Engström et al. [3] conducted a SLR, bridging the gap between the creative and technical nature of video games. Mizutani et al. [20] performed a SLR on the usage of software architectures in game mechanics. Viana et al. [21] carried out a SLR on Software Engineering for Pervasive Games. Zhu and Wang [22] analyzed the state of the art of Model-Driven Game Development (MDGD) research with a literature review. Politowski et al. [23] performed a literature review and a survey of developers about the topic of automated video game testing. Osborne et al. [24] conducted a SLR about processes and management in GSE research. Vargas et al. [25] performed a mapping study to reveal the growth in the quality of serious games. Aleem et al. [26] carried out a SLR on the production life cycle of video games.

All of the previous academic literature reviews focus on specific topics within GSE. To the best of our knowledge, the only study that addresses software and its engineering in video games is the SLR published by Ampatzoglou and Stamelos in 2010 [5], which is highlighted in bold in Table 1.1. They confirmed that GSE is “*a fertile domain*” with high activity, intensity growth, and open avenues to continue and validate their study with future research.

This study consolidates the hypothesis given by Ampatzoglou and Stamelos [5], providing an updated perspective on the state and evolution of GSE. The results of their study and ours are compared, rendering the state of GSE from its beginning to the present. The difference between our study and previous research is that our study explores the last decade of GSE, which has never been researched before, and it also focuses on industry-scale evidence.

1.1.3 Structure of this work

The remainder of this study report is structured as follows: Section 1.2 displays some background on video games as software and presents the previous related work. Section 1.3 addresses the research questions and the study design, explaining the review method step by step as we performed it. Section 1.4 presents the results we obtained as the data extracted from the reviewed studies answering the research questions. Section 1.5 presents a discussion of those results. Section 1.6 looks forward into the future of GSE. Section 1.7 presents possible threats to validity, limitations, and a quality assessment of this review. Finally, Section 1.8 concludes the study report and identifies the next steps for future work.

1.2 Background

Video games are software and, thus, present similarities with methods and techniques that are present in Software Engineering. Nevertheless, video games are a unique piece of software in terms of size, complexity, and creativity, constituting plenty of differences between GSE and traditional SE. As we performed the literature review, we discovered how the research community earmarked efforts to determine the nature of video games as software, finding their similarities and differences with traditional software.

We performed a manual search of the research in order to find the similarities and differences between these two fields. We found that there are four main similarities shared between GSE and traditional SE:

- The design decisions are based on a commercial perspective, with priority on the business side [7, 26, 27, 28, 29].
- Technical work is similar between the fields. Even if the requirements are specific for each domain, the programming work must adhere to those requirements [2, 7, 27].
- Both face similar problems in project scope, dropping features, and design. [7, 10, 27, 28, 30, 31].

- Despite potential disparities due to team size and role heterogeneity, project management exhibits similarity between the two fields, as game development teams have adopted agile methodologies from traditional software engineering [3, 4, 7, 31, 32].

On the other hand, the differences we found have a critical impact on the approach of video games towards software. The studies concerning GSE that tackle the differences have ten points in common:

- The teams developing a video game are multidisciplinary, with non-engineer heterogeneous roles (e.g., art, music, animation, or marketing) that must be integrated into the work pipeline [31, 6, 8, 3, 12, 33, 18, 26, 10].
- The process of game development is highly iterative, and the design is made while implementing the software itself [31, 7, 4, 3, 12]. This leads to a dearth of non-agile methodologies for video game development [6, 8, 11, 18].
- Video game developers usually have fixed deadlines in a market where delaying a release can impact the business considerably [5, 12].
- Video game developers must manage a large number of assets since customers expect large amounts of content, mirroring the industry's progression from smaller, technologically limited games to the current trend of expansive, content-rich gaming experiences [31, 6, 7].
- Video games also place emphasis on player experience testing and game-specific testing. This emphasis on player experience takes into account the interconnection of aesthetics, interactive mechanics, and storytelling. Game-specific testing encompasses comprehensive gameplay evaluation, including controls, mechanics, and level design assessments, such as balancing. Not only must game developers perform technical testing, but they also need to test the artistic nature of their products. [6, 7, 33, 9].

- Post-release features and content addition are standard practices in the video game industry. The product's life cycle can be extended years after launch [7, 33].
- Video games have non-functional endemic requirements that are highly subjective and more focused on player experience [33, 7, 32, 26, 10]. These user-driven requirements cannot be described technically since they are based on the purely subjective opinion of the users and focus on abstract concepts such as "*fun*" [4], "*entertainment*", "*aesthetics*", "*flow*" [32], or "*creativity*" [33] instead of being based on real-world and measurable concepts as in traditional software [7].
- Video games require the management of complex software systems that must communicate and synchronize with each other for an understandable and enjoyable experience. These systems include 2D & 3D rendering, physics, sound, and artificial intelligence [7, 12, 26].
- The complexity of video games usually forces game developers to use sophisticated tools called "game engines" that require specific expertise. These engines require skills other than software frameworks, such as art, design, or musical skills. These engines are also used differently than frameworks. To ease the design and implementation process, the game developer often uses scripting languages or domain-specific modeling languages that are different from the language the engine is coded [7, 2, 8, 32].

While GSE shares some traditional practices with SE, many other practices differ, and there is a need for specificity while dealing with video game software. These facts bring us to the same conclusion that many other researchers have come to previously [6, 7, 4]: *Game Software Engineering is a scientific domain rather than a soft skill topic with only a peripheral research activity* [5].

1.3 Review Method

The systematic review process details how to analyze and synthesize the evidence addressed by other scientific studies related to the subject in an unbiased and repeatable way. We followed the guidelines given by Kitchenham and Charters [34]. The stages of a systematic review are divided into three main phases: Planning the review, Conducting the review, and Reporting the review [34, 35]. Every phase is a combination of other more straightforward procedures: The Planning phase combines the specification of research questions and the development and validation of the review protocol. The Conducting phase requires the identification of relevant research, the selection of the primary studies, the assessment of the studies' quality, the extraction of the required data, and the synthesis of the data. Finally, the Reporting phase is performed by writing the review report and validating it. Following the best practices [36, 5] to plan our review, we define the following six elements: research questions, search process, inclusion and exclusion criteria, quality assessment, data collection, and data analysis.

The first and second authors participated in every step of the review process, double-checking every step to ensure the accuracy of the inclusion and exclusion processes and the quality assessment. When disagreements occurred, the two reviewers discussed the possibilities. If both reviewers did not reach a consensus, the other authors resolved the dichotomy.

1.3.1 Research Questions

Alves et al. [37] use a classification approach for the maturity of a particular method or tool. This is a revision of Kitchenham's levels of study design in software engineering based on the evidence hierarchy [38]. This classification's strongest ranks are *evidence obtained from industrial studies* and *evidence obtained from industrial practice*. The rating "*industrial practice*" indicates that the method has already been approved and adopted by some organizations. Such daily engineering practice provides convincing proof of the study's validity [37]. By selecting only studies with industrial relations, we ensure that the quality of our provided results matches the high standards of the industry.

This research field is growing actively year by year, and thus in quality and maturity [5]. Because of that, we have the opportunity to specify the systematic review to focus on software engineering for **industry-scale** computer games. The term “industry-scale computer game” refers to games and tools developed in industrial practice or for commercial purposes, thus, we do not consider academic prototypes as industry-scale computer games. This study addresses similar research questions (RQ1, RQ2, RQ3, and RQ4) from Ampatzoglou and Stamelos’ [5] study with adaptations to our more restrictive research domain to allow for a more straightforward comparison.

- RQ1:** Has the intensity of the research activity on software engineering for industry-scale computer game development changed over the years?
- RQ2:** How have software engineering research topics in the domain of industry-scale computer games changed over the years?
- RQ3:** How have research approaches in the domain of industry-scale computer games changed over the years?
- RQ4:** How have empirical research methods in the domain of industry-scale computer games changed over the years?
- RQ5:** What is the quality of industry-scale computer game research currently?

RQ1 shows the existence of an increase or a decrease in research activity. We compared the variation for research in software engineering in computer games since 2009 to the corresponding variation of the years prior to 2009, as provided by [5].

To address RQ2, RQ3, and RQ4, we used the data gathered in the review and also compared it with the current research study on the topic by Ampatzoglou and Stamelos [5] prior to 2009. We associated each primary study with a research topic, a specific research approach, and a specific research empirical methodology, similar to what Ampatzoglou and Stamelos did. For each RQ, we add a discussion drawing conclusions from the additional evidence obtained, shedding some light on the current state of GSE and identifying future trends.

With regard to RQ2, the classification system of the primary studies that [5] employed is the ACM Computing Classification System, which was widely adopted in computing-related engineering research. The ACM CCS was updated in 2012 [39], replacing the previous 1998 version. We classified the primary studies with this new version, but we also classified them with the 1998 version in order to make the comparison. These classifications are further explained in Section 1.3.4.

With regard to RQ3, scientific research studies can be classified with respect to their approach [40]. According to the classification scheme proposed by R.L. Glass et al. [40] and later on used by Ampatzoglou and Stamelos [5], scientific studies can be categorized by the overall approach undertaken in performing the research. The main scientific approach categories are descriptive, exploratory, and empirical approaches [5]. Descriptive studies describe a system, tool, or method. Moreover, SLRs are considered to be descriptive. Exploratory studies determine the best research design, data collection method, and selection of subjects. The findings of empirical studies are based on direct or indirect observation of real subjects.

With regard to RQ4, empirical research studies can be classified with respect to the method of empirical investigation used to evaluate new techniques, methods, and tools [41]. These method categories are surveys, case studies, and experiments. Experiments have a set of subjects performing a task in a highly controlled environment. The results are obtained by observing the subjects during the experiment, inspecting the task outcome, or questioning the subjects after performing the task. Surveys are usually used if the method under study has been in use for a while. In surveys, a set of subjects is asked to fill in questionnaires. The results are obtained with valid answers to these questionnaires. Finally, case studies study a methodology by monitoring a project, activity, or assignment with respect to it. The results are obtained with project measurements [42].

The same classification of approaches and empirical methods was employed in [5], so we can compare the results in that study with our own and see an evolution of the results before and after 2009.

RQ5 allows us to identify the common problems in current research that are tampering with the quality of the studies. This will help researchers to avoid those issues in future work in the field of GSE.

1.3.2 Literature Search Strategy

We performed a database search to gather all of the studies that were relevant to the research questions. To do this, we performed the following steps: search term extraction, database selection, and selection process.

Search terms

We used PICO (Population, Intervention, Comparison, Outcome) as suggested by Kitchenham and Charters [34] to find the search terms for the query.

Population. In software engineering, population refers to the application area in software engineering [34]. For our study, this is computer games.

Intervention. In software engineering, intervention refers to the methodologies, tools, technologies, or procedures that are used to create the software [34]. In our study, this corresponds to 'engineering', 'development', 'requirements', 'design', 'coding', 'testing', 'verification', 'evolution', and 'maintenance'. We have extracted these terms to describe the different software engineering activities, comparable to how Mao et al. [43] extracted the terms for a similar survey reviewing another broad SE field.

Comparison. In software engineering, comparison refers to the methodologies, tools, technologies, or procedures being compared with the intervention [34]. In our study, comparison is not applicable.

Outcome. In software engineering, outcome refers to factors that should be important to practitioners [34]. In our study, the outcome includes approaches that focus on industry-scale video games and Software Engineering. We decided to perform this step manually as part of the inclusion and exclusion criteria to avoid losing valuable work if it is filtered automatically. There are many studies performed in industry-grade video

games where the word “industrial” does not appear in the whole study report.

After recognizing the search terms from the research questions, these terms need modifications and additions to be effective in a search string.

The term *computer game* refers to electronic games that are developed to be executed on a computer. These games can be nominated as *video games*, an alternative synonym that refers to games that are developed to be executed on any device that supports them, including computers. Even though it is a more generic term, we included it to extract studies deemed significant. Since we only intended to review computer games, we make up for this addition with the selection criteria, which is detailed in Section 1.3.2. With this criteria, we exclude any study about non-computer games. *Video games* can also be spelled as *videogames*. Additionally, some databases need the distinction between plural and singular words, so we added these variations to the search string.

Our Intervention terms can also refer to ideas that are too broad and can depict concepts outside of the SE field, so we added the word *software* before each term. For example, the term *testing* can be used in the medical field, and, thus, it is changed to *software testing* to ensure its relevance to SE.

For the search string, we used the Boolean operator OR to combine the terms inside each criterion and the Boolean operator AND between Population and Intervention. The terms of Population must appear in the title, abstract, or keywords of the searched studies to ensure the focus on computer game development. The terms of Intervention must appear anywhere in the studies found. The resulting search string is as follows:

TITLE-ABS-KEY (“computer game” OR “computer games” OR “video game” OR “video games” OR “videogame” OR “videogames”) AND ALL (“software engineering” OR “software development” OR “software requirements” OR “software design” OR “software coding” OR “software testing” OR “software verification” OR “software evolution” OR “software maintenance”)

Databases

We have considered the Scopus, Web of Science, IEEE, and ACM databases for this survey. We chose these based on the prior experience of Dyba et al. [44] and Kitchenham and Brereton [45]. These studies stated that using IEEE and ACM as well as two indexing databases is appropriate [46]. During the search, we considered all of the full years from 2009 to 2021 (both included). Figure 1.1 shows the number of search results per database.

Search process

Both reviewers followed the review and selection process depicted in Figure 1.1 using the 3,491 total results from the search in the four databases. To achieve the selection of studies for data extraction, the process performed had the following steps:

1. The reviewers searched four databases using the search string to collect the primary studies. We show the search results from each database in Figure 1.1.
2. The total number of 3,491 studies was filtered to exclude 517 duplicate records, decreasing it to 2,974 studies.
3. The non-duplicated studies were filtered, and we excluded all non-relevant studies for this study. For the fulfilment of this step, we applied the exclusion criteria considering each study's title and keywords. The selection decreased to 614 studies, which we selected for the following step.
4. We filtered the 614 studies. This time, we filtered the studies by abstract with the same exclusion criteria, and 362 were eliminated. The selection decreased to 252 studies not meeting any exclusion criterion.
5. We also filtered this selection of 252 studies by their relevance in answering the research questions, applying the inclusion criteria to the full text of each selected study. The selection again decreased to a final selection of 90 studies.

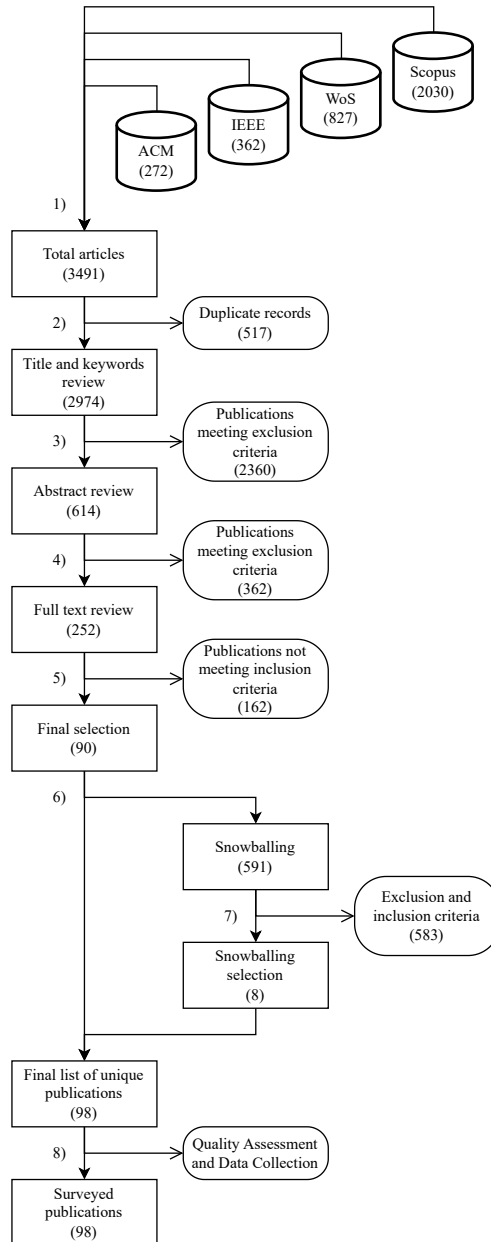


Figure 1.1: Overview of the search process

6. We selected all of the references from the selection of 90 studies for the application of backward snowballing [47]. We performed snowballing to search for possible missing studies. The initial list of snowballing studies counted was 591 studies.
7. We filtered the 591 studies in the initial snowballing list by applying exclusion and inclusion criteria to their title and keywords, abstract, and full text, with eight studies meeting all of the inclusion criteria and not meeting any exclusion criterion. We added these eight studies to the previous final selection.
8. With the final list of 98 unique publications, we performed a quality assessment and data collection of every primary study in the final selection to collect information that was suitable to answer our research questions.

As a result of this process, we confirmed all 98 selected studies as final sources. It is relevant to note that we could not retrieve three full-text study reports from any digital library. We contacted the authors, but only one of them answered. Thus there were two studies out of the 252 in the full-text review that we could not review.

The following subsections explain the inclusion/exclusion criteria, the quality assessment process, and the data extraction process.

Selection criteria

To answer our research questions, we evaluated the retrieved studies according to the following exclusion and inclusion criteria. All of the studies in our review had to be relevant to Software Engineering (classified under a SE topic) and focus on computer games. We discarded all of the studies that met any exclusion criteria from our study. In contrast, the studies that met all of the inclusion criteria were the ones selected for our study. Our process had six stages of filtering using the exclusion or inclusion criteria: identifying all relevant studies (3,491), removing duplicate records (517), excluding publications by title (2,974), excluding publications by abstract (362), excluding publications by full text (162), and excluding publications from the snowballing list (582).

The following criteria state when we excluded a study from our study:

- Studies that were duplicates of other studies.
- Studies that were not written in English.
- Studies that were not published online between 2009 to 2021.
- Studies presenting non-peer-reviewed material.
- Studies presenting peer-reviewed but not published in journals, conferences, or workshops.
- Studies that were summaries of conferences/editorials.
- Non-primary studies.

The following criteria state when we included a study in the study:

- Studies that were not focused on the social and educational impact of video games, such as serious games.
- Studies that were not focused on Artificial Intelligence (AI).
- Studies that were not focused on Content Creation.
- Studies that were in the field of Software Engineering.
- Studies that were focused on software engineering applied to industry-scale computer games development.

During this process, we found several interesting studies which we decided to exclude for not being closely related to computer games or software engineering issues. The main focus of these studies is on “artificial intelligence”, “mobile gaming”, virtual reality”, “augmented reality”, “content creation”, and “serious games”. The term *serious game*, as explained by Cheng et al. [48], “*has become a popular and particular term referring to any kind of video game-based learning and training (e.g., business, military, medical, marketing) or the so-called ‘edutainment’*”. We did not include studies about serious games because these studies focus on the health, social, or educational implications of games and not on the software that runs them or the engineering methodologies used to develop

them. Similarly, studies on “content creation” and “artificial intelligence” for video games are more focused on the content itself and applications rather than the SE behind them [49]. It is also important to note that we only considered full primary studies from journals, conferences, and workshops for the review. We have removed other studies, such as posters, summaries, and non-peer-reviewed studies, from the survey.

Since the selection process is not affected by the order of execution of the selection criteria, we present the criteria in no particular order. Figure 1.1 shows the entire process that we followed with the number of excluded studies and included studies.

The final selection consisted of ninety-eight (98) studies. These studies are presented in the *Studies Included in the Review* section at the end of the study report along with the complete data-set extracted from the study in the APPENDIX.

1.3.3 Quality Assessment

The assessment of the quality of primary studies is critical in a SLR [34] since it helps validate that the methodology and results of these studies are solid. Similarly to Ampatzoglou et al. [5], the quality of the published studies is satisfactory because of the high standards of the publishing process of the selected journals, conferences, and workshops. That is why our quality criteria were not used for filtering purposes to include or exclude primary studies in the review. As Kitchenham et al. [50] proposed, we assessed primary studies using questions. We show these questions in Table 1.2. Similar to Galster et al. [51] and Ali et al. [52], we adopted the quality assessment instrument used by Dyba et al. [53], and we decided to use a three-point scale to answer each question, either as “Yes”, “To Some Extent”, or “No”. Using a three-point scale, we avoided neglecting statements where authors provided only limited information to answer the assessment questions [52]. We assigned each quality assessment question a numerical value as answer (“Yes” = 1, “To Some Extent” = 0.5, and “No” = 0). A quality score was given to a study by summing up the scores for all of the questions asked [52]. We provide the results of the quality assessments in Figure 1.7. However, as Table 1.2 shows, these

scores refer to the quality of the reporting of a study rather than its actual quality [51]. As we explained in Section 1.3.1, by selecting only studies with industrial relations, we ensure that the actual quality of the results matches the high standards of the industry.

Because our questions for this quality assessment (shown in Table 1.2) are the same as the ones used in [51], we also followed their approach by applying the quality criteria to all of the studies included in the review rather than only to studies that report empirical methods.

Table 1.2: Questions to Assess Study Quality

N	Question
Q1	Is there a rationale provided for why the study was undertaken?
Q2	Is there an adequate description of the context (e.g., industry, laboratory setting, products used, etc.) in which the research was carried out?
Q3	Is there a justification and description for the research design?
Q4	Is there a clear statement of the findings, including data that supports findings?
Q5	Did the researcher(s) critically examine their own role, potential bias, and influence during the study?
Q6	Are limitations and credibility of the study discussed explicitly?

1.3.4 Data Extraction

We show the information collected from the studies in Table 1.3. This data is tabulated and statistically analyzed to investigate the intensity changes of the research activity on the topic, the most active journals and conferences in GSE research, the SE topics currently addressed in GSE research and the number of studies employing each research approach and

method. We collected the data for F1 to F7 from the meta-information of study reports. We collected F1, F3, F6, and F7 to keep that information of the study reports. F2 records the year of publication and helps to answer all of our RQs as they are time-related. F4 and F5 help manifest the most active research publication outlets and publishers for GSE studies. F8 records the quality score for each study. F9 records software engineering research topics based on the ACM classifications of 1998 and 2012. F10 and F11 record each study's research approach and method based on the selection criteria introduced in Section 1.3.1. F12 records the limitations or threats addressed by the authors of each study.

Table 1.3: Data collection form

N	Field	Research question
F1	Author(s)	n/a
F2	Year	RQ1, RQ2, RQ3, RQ4
F3	Title	n/a
F4	Venue (journal or conference)	RQ1
F5	Publisher	RQ1
F6	Keywords	n/a
F7	Abstract	n/a
F8	Quality score	RQ5
F9	SE research topic	RQ2
F10	Research approach	RQ3
F11	Empirical research method	RQ4
F12	Limitations	RQ5

Classifying topics

This section clarifies the classification of each study concerning the software engineering issue it involves. The classification system employed is the ACM Computing Classification System (CCS). In 2012, ACM undertook a major revision of the CCS, but we classified the primary studies using the 1998 version in order to compare our study with Ampatzoglou and Stamelos' study [5]. We also included the 2012 version classification

for two main reasons: to check how appropriate it is for GSE classification and to be able to compare future work to ours by using the newest version of this classification system if needed. The topics used for the 1998 ACM CCS version are shown in Table 1.4, and the topics used for the 2012 ACM CCS version are shown in Table 1.5.

The topics shown for either classification system are not the complete list of categories. Each one of the topics encompasses more subtopics. We used the classification systems with that depth of detail in order to ensure that we could group the results into representative topics.

Both reviewers performed the primary study classification for both CCS versions while performing the full-text review. For the classification process, the authors checked if the study under review was already classified under one of the two versions of the CCS. We reviewed studies dating from 2009 to 2021; therefore, some of them still used the 1998 version instead of the 2012 version. If the study was already classified with either of the two CCS versions, the author currently reviewing the study maintained that topic and classified it for the other version of the CCS. The author did this for the two versions if it was not classified using any CCS version. When the two authors reviewed and classified the study.

It is also worth noting that all software testing and usability testing are included in the topic *5.3.4 Software verification and validation* for the 2012 version of the CCS since they did not match well in any other topic.

1.4 Results

This section shows the data extracted from the Literature Review of the 98 studies concerning the research questions specified in Section 1.3.1. Additionally, the data is compared with the research activity of the previous decades. To compare with the data until 2009, we consider the study by Ampatzoglou and Stamelos [5], but our selection criteria are more restrictive because our study focuses on industry-scale computer games. In order to address this issue, we have reviewed how many studies of the final selection in Ampatzoglou and Stamelos' study [5] match our criteria. The previous SLR identified 84 studies from 1994

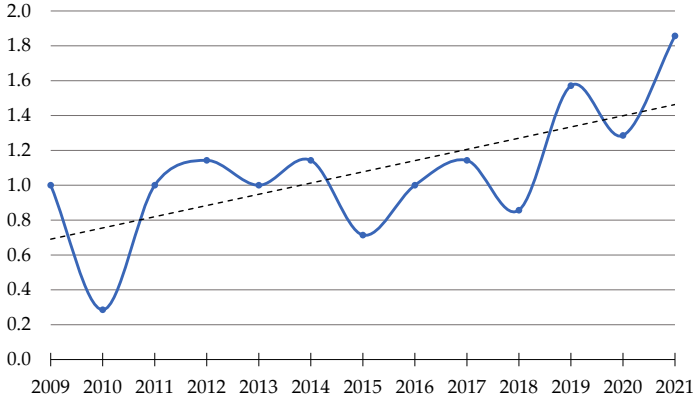


Figure 1.2: Increase in GSE Research Activity

until 2009 (15 years). Of those studies, only 24 are industry-scale. We show these studies in the APPENDIX. The number indicates the position on the list of studies provided by their authors. Our SLR found 98 industry-scale studies from 2009 to 2021 (12 years). The data-set of the whole search process, selection process, data collection, and quality criteria are available at <https://svit.usj.es/SLR-GSE-dataset/> and at <https://doi.org/10.5281/zenodo.8242657>.

1.4.1 RQ1 Research Intensity

The most straightforward comparison to address the evolution of the research intensity is counting the final number of studies on this topic before and after 2009. There are 24 studies before 2009 to compare to our data-set of 98 studies after 2009. There has been over four times more industry-scale research from 2009 to 2021 than before 2009. Furthermore, the time interval in which we conducted our study is three years shorter.

Figure 1.2 shows an increase in the research intensity within our time frame. It shows the number of publications relative to the first year of the time frame, 2009. In addition to the increase compared to previous years, GSE research has kept increasing until now and shows a tendency to keep growing in the future.

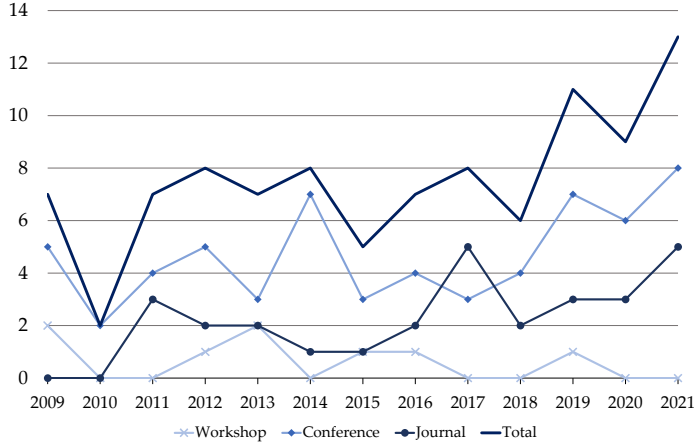


Figure 1.3: Research activity per year and publication type

Figure 1.3 shows the distribution of publications per year divided by type of publication: Journal (29), Conference (61), or Workshop (8). Most of the studies are published in conferences, followed by journals, and lastly, workshops. We found 86 different venues of publication for the 98 studies.

1.4.2 RQ2 Research Topics

Table 1.4 and Figure 1.4 show the comparison between the topics of our data-set and the previous data-set [5] filtered by our selection criteria excluding all non industry-scale. It is worth mentioning that every single topic has been researched more after 2009, with the exceptions of *D.2.0 General*, with the same number of studies found (two), and *D.2.1 Requirements/Specifications* with only one study less (from nine to eight).

With the increase in research intensity and maturity, the field has also evolved to be more evenly distributed among the different research topics, as shown in Table 1.4 and Figure 1.4. The main topic before 2009 was *D.2.1 Requirements/Specification*. After 2009, we find studies on every topic categorized in the 1998 version of the ACM CCS, emphasizing topics *D.2.2 Design Tools and Techniques*, *D.2.8 Metrics*, and *D.2.11 Software Architecture*. These are the three topics with the biggest increase in interest by the community and part of the five topics most studied,

Table 1.4: GSE Research Topics CCS '98

Software Engineering Topic	Before 2009	After 2009
D.2.0 General	2 8.33%	2 2.02%
D.2.1 Requirements/Specifications	9 37.50%	8 8.08%
D.2.2 Design Tools and Techniques	0 0.00%	15 15.15%
D.2.3 Coding Tools and Techniques	3 12.50%	6 6.06%
D.2.4 Software/Program Verification	0 0.00%	6 6.06%
D.2.5 Testing and Debugging	3 12.50%	12 12.12%
D.2.6 Programming Environments	2 8.33%	3 3.03%
D.2.7 Distribution, Maintenance and Enhancement	0 0.00%	3 3.03%
D.2.8 Metrics	0 0.00%	10 10.10%
D.2.9 Management	5 20.83%	12 12.24%
D.2.10 Design	0 0.00%	5 5.05%
D.2.11 Software Architecture	0 0.00%	10 10.10%
D.2.12 Interoperability	0 0.00%	2 2.02%
D.2.13 Software Reuse	0 0.00%	4 4.04%

along with *D.2.5 Testing and Debugging* and *D.2.9 Management* after 2009.

Table 1.5 shows how the 2012 version of the ACM CCS topics does not represent GSE as its previous version. In contrast to its previous version, the 2012 one shows the studies as unevenly distributed, with empty topics and a small number of topics that are very populated.

Figure 1.5 shows the timeline of publications per research topic. In order to make this chart more legible, we have grouped the years into four sets of three years except for the last one, which contains four years. For the same reason, we collapsed the fourteen CCS '98 topics into six broader topics, as Ampatzoglou and Stamelos did [5].

It can be observed that more than 70% of the research has been done in the second half of the time frame for *General*, *Management*, and *Quality*. *Implementation* and *Design* have continued to grow in the last few years with more than 65% and 50% publications, respectively. The only exception is *Analysis*, which presents more intensity in the first half of the time frame.

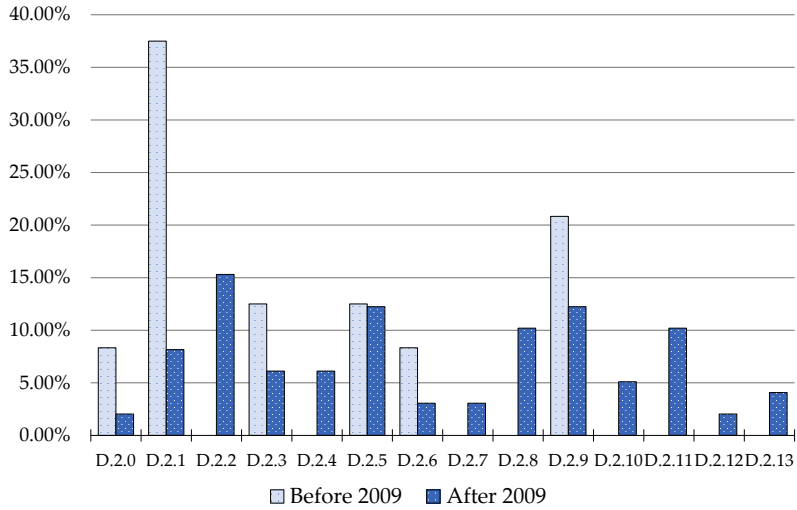


Figure 1.4: Research Topics

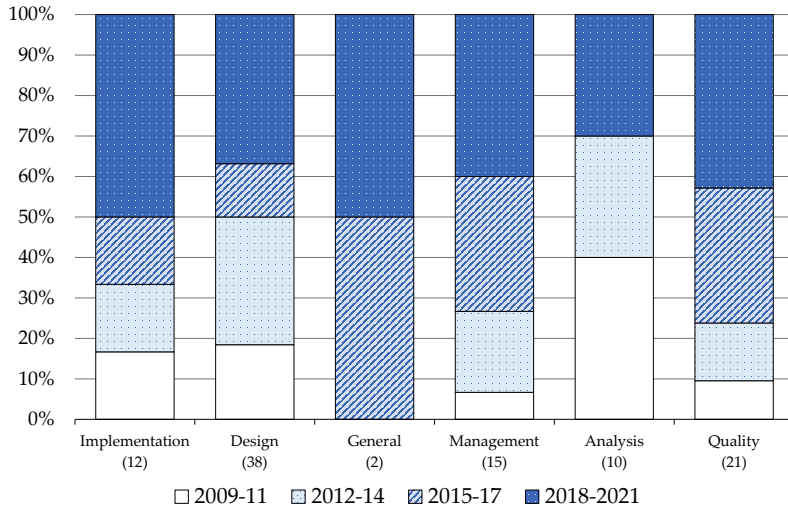


Figure 1.5: Research Topics Growth Timeline

Table 1.5: GSE Research Topics CCS '12

Software Engineering Topic	#	%
5.1.1 Contextual software domains	4	4.04%
5.1.2 Software system structures	14	14.14%
5.1.3 Software functional properties	2	2.02%
5.1.4 Extra-functional properties	3	3.03%
5.2.1 General programming languages	0	0.00%
5.2.2 Formal language definitions	1	1.01%
5.2.3 Compilers	0	0.00%
5.2.4 Context specific languages	3	3.03%
5.2.5 System description languages	2	2.02%
5.2.6 Development frameworks and environments	3	3.03%
5.2.7 Software configuration management and version control systems	0	0.00%
5.2.8 Software libraries and repositories	0	0.00%
5.2.9 Software maintenance tools	0	0.00%
5.3.1 Designing software	14	14.14%
5.3.2 Software development process management	13	13.27%
5.3.3 Software development techniques	9	9.09%
5.3.4 Software verification and validation	26	26.26%
5.3.5 Software post-development issues	4	4.04%
5.3.6 Collaboration in software development	0	0.00%
5.3.7 Search-based software engineering	0	0.00%

The rest of this section features the description of primary studies to represent the state of the art of GSE. We grouped these descriptions by the Research Topics CCS '98. Each description has a citation to the list of studies included in this SLR. It is relevant to note that we separated the list of the review studies from the list of references, so the citation to the review studies can be distinguished by the *S*- prefix. Additionally, the reader can find a table filled with the full study data-set in the APPENDIX.

General

In two studies, [S1, S2], the authors test the security of industry-scale online video games.

Requirements/Specifications

In two studies [S3, S4], the authors propose that semantic models can be beneficial for the design and development of video games. In [S5, S6], the authors explore the definition of video games as multi-agent systems. In [S7], the authors explore how requirements engineering is used in the video game industry.

In [S8], the authors perform a qualitative study regarding the tool infrastructure of game industry developers.

Two studies [S9, S10] interview game developers to illustrate the experiences of trying to make accessible games and how they manage requirements engineering, respectively.

Design Tools and Techniques

Two studies highlight the importance of analyzing the experience that players have with video games in a pragmatic and hedonic way (from a cultural and emotional perspective) [S11, S12].

In two studies [S13, S14], the authors focus on accessibility by proposing eye-gaze interaction systems. Two studies [S15, S16] focus on improving the User Interface (UI) and reviewing its relationship with the game world. Another study [S17] interviews game developers to illustrate the most widely used practices for achieving an optimal user experience. In [S18], the authors presented a usability inspection method that can be used in the early stages of Model-Driven video game development.

Furthermore, two studies propose the usage of Model-Driven Development in video games, comparing it to Code-centric Development [S19] or using it with Software Product Lines (SPL) [S20]. In four studies [S21, S22, S23, S24], the authors investigate modelling in video games, focus-

ing on the design and its analysis. Finally, in [S25], the authors examine the potential of using Text-To-Speech (TTS) to prototype voice acting in game development.

Coding Tools and Techniques

In [S26], the authors focus on the video game domain to explain a technique of parallel software that is focused on data movement. Four studies deal with rendering [S27, S28, S29, S30], explaining illumination algorithms and 3D rendering effects used in the industry. These provide a technique for increasing modularity and encapsulation of shaders in real-time graphics APIs, and thus explore which rendering technique provides the most attractive results for the users.

Additionally, in [S31], the authors propose a framework and methodology for optimising the building of systems, for example this framework can be used for building shader systems.

Software/Program Verification

Two studies deal with mathematical models for verifying and validating video game gameplay and logic [S32, S33]. In three studies, the authors create frameworks for analysis and evaluation, focusing on Real-time Strategy (RTS) games [S34], the usability of video games developed with Model-Driven Development [S35], and Unreal Engine 4 blueprints [S36]. In [S37], the authors detect hot spots in the code of video games.

Testing and Debugging

Two studies explore new testing methods, where the authors present an incremental method for extracting adaptive tests from formal, object-oriented specifications [S38] and use combinatorial testing to establish a framework to improve the testing efficiency of video games [S39]. Another two studies describe approaches for automatic verification and testing in node-based visual script notation [S40] and code [S41].

Furthermore, one of the most important tasks of video games is to provide quality entertainment to their players, so they need a significant amount of testing on user-centered issues, usually through playtesting. In two studies, experiences and methods of playtesting are described and performed for independent video game development studios (*indie* studios) [S42, S43]. The second study also explores the differences between the highlighted features in game review analysis and in play-test reporting. In [S44], the authors explore the usage of players' game reviews as a cheap and effective way of collecting user feedback.

Moreover, crashes in video games significantly produce a negative user experience, so a new crash reporting system is proposed in [S45], improving data reliability, data availability, and query responsiveness while reducing CPU and memory usage. In [S46], the authors investigate how to collect and centralize events with a distributed system for debugging in online video games. In addition, in [S47], the authors propose a taxonomy for bug types after analyzing common existing bugs in industrial games, and, in [S48], the authors focus on fixing bugs at runtime.

Finally, in [S49], the authors study the differences and similarities between software and game industries on testing and quality assurance.

Programming Environments

In [S50], the authors present an engine is presented focused on creating visual novels. Another study compares open source game engines with traditional software frameworks finding differences [S51]. Finally, [S52] presents a tool to ease communication among game development departments.

Distribution, Maintenance and Enhancement

In [S53], the authors explore how Lehman's laws of software evolution are applied in video games. In two more studies [S54, S55], the differences between traditional software and video games patches are highlighted, providing an extensive analysis of the impact on the users after every patch in an industry-scale competitive online video game and how these affect players' engagement.

Metrics

In four studies [S56, S57, S58, S59], the authors investigate the capture of metrics. Furthermore, three studies [S60, S61, S62] present how to analyze metrics and telemetry data based on the player's gameplay for the improvement of the next video game titles or pattern generation. Finally, in [S63, S64, S65], the authors present the definition and usage of new and adapted usability heuristics.

Management

In [S66], the authors introduce a maturity model that focuses on evaluating video game development methodologies in an organization. Also, in [S67], the authors attempt to adapt the ISO/IEC 29110 to be applicable to the video game industry, concluding that very small companies could benefit from it. Furthermore, some studies investigate how traditional Software Engineering management techniques could be applied to video game development [S68, S69] or compare them with techniques and frameworks that are already being used in video game development [S70]. Some studies explore how development life cycle models could be used in video games [S71], what framework video game development processes should follow [S72], and how to manage and improve communication among the members of a development team [S73].

In [S74], the authors present a survey exploring risk management in the video games industry. In [S75], the authors also present a survey along with postmortem examples to study how crunch affects the industry and its employees. In one study [S76], the authors present a survey to pro-

fessional game developers about technical debt, a problem that they are aware of for which almost no solution has been implemented. Finally, [S77] presents a survey dealing with how video game companies are using SCRUM sprints.

Design

Three studies concern the Game Design Document (GDD), reviewing how GDDs are written in the industry with no standard and proposing new guides for building them [S78, S79, S80]. In [S81], the authors use a gameplay data analysis to fix and improve game design. In [S82], modding is described as a common practice among video game communities where open-source video game software is customized.

Software Architecture

With regard to the network architecture in online games, in [S83], the authors propose a software architecture for traffic generation in massively multiplayer online role-playing games (MMORPGs), whereas in [S84] the authors analyze networking approaches of online video games and exploitable security concerns. In [S85], the authors introduce a high-performance and robust rendering pipeline. In [S86], a survey deals with the differences in the usage of architectures between traditional software and video game development and its evolution in the last years.

In addition, in two studies [S87, S88], the authors present an evaluation of a simple method to find architectural problems in a product line of computer games that use the Model-View-Controller (MVC) pattern, and they apply the MVC pattern in video game development to prove that it solves common problems. [S89] employs multi-threading in a video game engine to demonstrate that the performance is improved. In [S90], the authors describe an architecture for economy systems in video games that reduces their implementation and maintenance costs. In [S91], the authors theorize about formal models being useful for video game development. In [S92], the authors explore how the multiple platforms in the market affect both software and video game development.

Interoperability

In two studies [S93, S94], the authors explain how they created middle-ware for multiplayer games.

Software Reuse

In [S95] presents an engine allowing generic platform-independent video game creation. Two studies [S96, S97] deal with DSLs for code generation at runtime for the video game domain.

Finally, in [S98], the authors focus on code transpiling to solve the issue of changing a game engine that is currently in use while in development.

1.4.3 RQ3 Research Approaches

GSE practitioners have shifted their research approach drastically in the last few years, as shown in Figure 1.6. Currently, there is almost a complete lack of Exploratory studies, decreasing from more than 40% to 1% of the studies. In contrast, empirical methods have more than doubled in interest in the last few years. It is worth noting that, in the last decade, exploratory papers are almost non-existent for industry-scale computer games.

1.4.4 RQ4 Empirical Methods

According to Figure 1.6, the case study was the least widely conducted empirical method by researchers before 2009, but it has evolved to be the most widely conducted method since 2009 in more than 50% of the empirical studies.

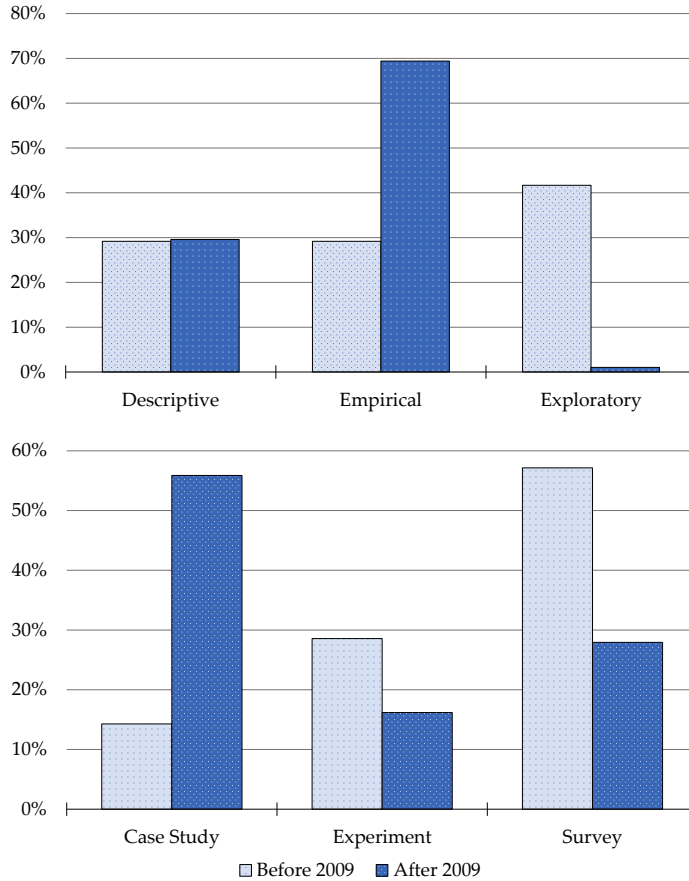


Figure 1.6: Research Approaches and Empirical Research Methods

1.4.5 RQ5 Research Quality

With regard to the quality questions elucidated in Section 1.3.3, we find that most studies got “Yes” or “To Some Extent” to the questions about motivation and context (Q1 & Q2). Figure 1.8 also reveals how the justification of the research design and the inclusion of data that support findings (Q3 & Q4) is less addressed by the community. Nevertheless, more than 70% of the studies tackle them, at least, to some extent. Finally, GSE researchers generally lack an explanation of the critical

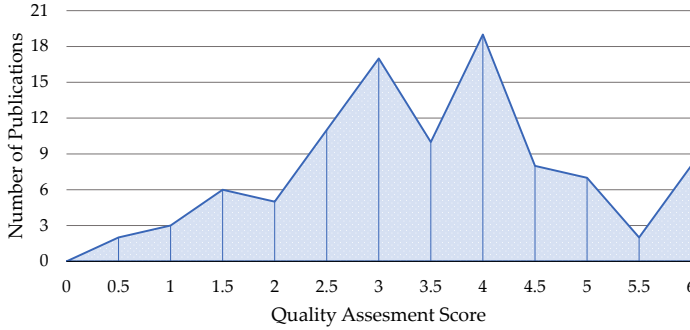


Figure 1.7: Publication count per quality total

examination of their own work, either of potential biases and influences affecting their study or the limitations of that work (Q5 & Q6).

There are 41 out of 98 studies that address (at least to some extent) the limitations of the study, and 13 out of 98 address their own potential bias or influence. The questions least addressed by the community (Q5 & Q6) generally comprise the section devoted to study limitations, threats to validity, or similar. This section only appears in 13 studies. The rest of the studies that acknowledge their study validity do it briefly inside other sections related to method, discussion, or conclusion. Even when the studies present threats to validity, 13 do not present mitigation strategies, another 13 studies perform the mitigation, and 15 intend to pursue it as future work.

It is worth noting that only 7 studies use best practices or widely accepted methods. The validation of other studies relies on performance evaluations or user acceptance by conducting questionnaires to practitioners in the industry.

1.5 Discussion

1.5.1 RQ1 Research Intensity

As games grow in size and complexity, developers need new techniques and approaches to manage their ambitious software. We observed in

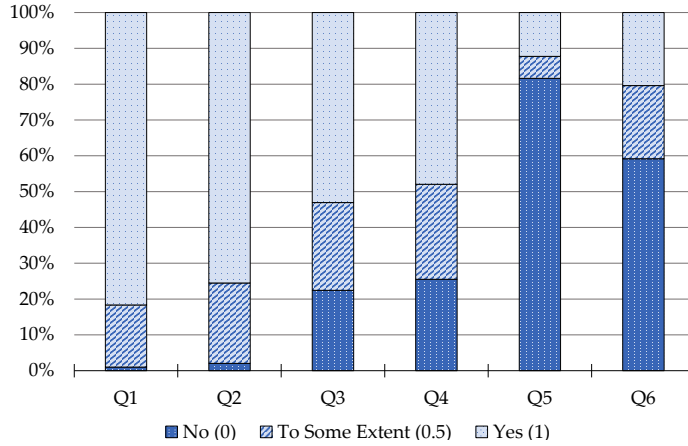


Figure 1.8: Research Quality

Section 1.4.1 that the number of studies on GSE has been growing over time and that interest in GSE research matches the continuous growth of the video game industry [1]. The GSE research community is responding by bringing innovation to them.

The 98 studies are dispersed among 86 different venues of publication. These results show that, while there are emerging venues out there that are focused on GSE research (such as IEEE Transactions on Games, Conference on Games, ICSE Workshop on Games and Software Engineering, or Workshop on Automated Software Engineering for Computer Games), the research community in this field is still scattered. The full list of publications by venue is displayed in the supplementary material.

It is worth noting that, even though there are no focused venues, these are linked to a few publishers. IEEE holds the majority of publications, with 40% of the publications. This is in contrast with the previous years, when ACM was the main publisher [5].

The growing body of research dedicated to Game Software Engineering (GSE) signifies a rising interest in enhancing the development and maintenance of video games. This surge in scholarly attention highlights the recognition of games as complex software systems that require specialized engineering practices to ensure their quality, stability, and efficient

production. As the video game industry continues to expand, the significance of GSE research becomes even more apparent, showcasing a direct response to the industry's demand for more effective tools and techniques.

1.5.2 RQ2 Research Topics

Before 2009, the main topic in GSE research was *D.2.1 Requirements/Specification*. It is usual for a young domain to attract researchers on this topic [5].

Video games are a special type of software with a major focus on user experience and interface, as video games require the player to perform a great range of actions while at the same time being enjoyable. At the same time, they are complex systems with many subsystems of high quality and a large amount of content that interact with each other. Some studies on *D.2.2 Design Tools and Techniques* focus on user experience and interface, while some of them focus on the documentation of the systems. Studies on *D.2.11 Software Architecture* focus on the architecture of those systems. These two topics tackle the challenges of size and complexity that characterize the high-standard industry of video games. This is probably a reason behind the increase in research on those two topics.

The increase of interest in *D.2.8 Metrics* can also be explained by the nature of this domain of software. These large systems, along with the high population of players, can provide plenty of data, which helps improve game experience and performance. This leads to researchers and industry developers growing interested in getting and managing heuristics from the software usage.

It is also worth noting that the interest in *D.2.5 Testing and Debugging* is similar before and after 2009. Following the previous reasoning, video games as software are big, with a high density of features that can increase unwanted behavior. In this field, there is also a difference between *software testing*, which focuses on errors in the logic of the systems, and *game-play testing*, which considers meeting the standards of quality and the player's experience. These standards deal with the correct enjoyment,

quality, and performance. The increase in interest in topic D.2.5 goes together with the high amount of effort and resources that the industry puts into testing [5].

We observed that *D.2.9 Management* is the second most studied topic. As seen in Section 1.2, development teams are bigger and more heterogeneous than traditional SE teams. The fact that practitioners and researchers are focusing on this topic means that managing large software and teams is still an ongoing problem that needs to be addressed because of the great rise in super productions of video games.

The studies in the 2012 version of the ACM CCS topics are very unevenly distributed, with empty topics and a small number of topics that are very populated. The best example is the topic *5.3.4 Software verification and validation*. Most of the studies that are categorized in three different topics from the 1998 CCS (*D.2.1 Requirements/Specification*, *D.2.5 Testing and Debugging*, and *D.2.8 Metrics*) are all collapsed inside the 5.3.4 topic from the 2012 CCS. Also, these three categories have proven to be of great interest to the community. The 2012 ACM CCS replaces the traditional 1998 version of the ACM CSS in serving as the de facto standard classification system for the computing field [39]. However, it does not render the current state of the GSE research domain correctly, making the differences between these two fields more evident.

We observed that research related to *Analysis* has more intensity from 2009 to 2014, while more than the 70% of the research related to *General*, *Management*, and *Quality* has been done from 2015 to 2021 and the intensity of the research related to *Implementation* and *Design* has continued to grow from 2018 to 2021. The video game industry experienced significant growth during the early 2010s, with the rise of mobile gaming [54] and indie game development [55]. This expansion may have led to an increasing volume of research on management for new emerging development teams of different sizes and on quality to meet the demands of a growing and diverse player base. In later years, as the market became more competitive, there could have been a greater emphasis on design and implementation aspects to stand out from the competition by offering a software aspect that demonstrates superior performance and scalability.

Finally, we observed some established and emerging trends in the reviewed studies, which are discussed further in Section 1.6:

- Model Driven Engineering appears in multiple studies on different aspects of GSE [S18, S19, S22, S35, S36].
- Software reuse appears to be a recurring practice and subject of research for both different products and different platforms [S95, S50, S98, S89, S92, S8, S20, S36, S92, S1].
- The approach reported in many studies is evaluated in a specific environment and it is not trivial to generalize it [S55, S54, S34, S62, S2, S13].
- The implementation of traditional software engineering methodologies in game development teams often proves ineffective [S66, S70, S71, S77, S75].
- Games as a Service (GaaS) are becoming increasingly prevalent in the gaming industry [S54, S55, S62, S2, S94, S93].
- Multiplayer online games create the demand for reliable, robust, and secure systems that operate without compromising performance [S93, S94, S83, S2].
- Many studies propose the use and improvement of Game Design Documents (GDDs) for better communication and documentation [S4, S5, S6, S78, S79, S80, S77]. However, an established standard for them is yet to be achieved and they do not seem to be a definitive solution.
- Among the requirements that need to be defined in video games, there are several differences compared to traditional software, and these requirements have not yet been addressed specifically [S7, S3, S4, S5, S6].
- Testing is another recurring topic in GSE since it is more challenging to automate than traditional software [S38, S39, S40, S41, S42, S43, S44, S46, S48, S49].

As the gaming landscape evolves, games are expanding in size and intricacy, necessitating new strategies and methodologies to manage the complexity that arises from their development effectively. Developers are confronted with the challenge of creating games that offer expansive worlds, interconnected mechanics, and rich narratives, which in turn demand innovative approaches to ensure smooth software development. This drive for innovation stems from the recognition that traditional software engineering practices might need to address the unique challenges presented by game development.

1.5.3 RQ3 Research Approaches

Exploratory studies are usually expected to be more prominent in a young research area and interest in this research approach decreases as the field develops [5]. This may be why it is the least exciting approach for practitioners. There is almost a complete lack of studies using this approach, decreasing from more than 40% to 1% in the last decade. This decrease may be a cause for the lack of quality in the report about the studies. We addressed this in Section 1.4.1, where the issues related to the validity of the research are poorly questioned.

Empirical methods have more than doubled in interest in the last few years. As GSE has grown in maturity, companies have noticed the benefits of collaborating with researchers and have started doing it in a more active way. This has increased the number of empirical studies and decreased the gap between industry and academic research. This results in practitioners presenting industry-scale demonstrations for their studies. The hypothesis of GSE growth in maturity is also supported by the results, which conclude that the field has also evolved to be more evenly distributed among the different research topics.

Along with the increase in intensity, the concurrent trajectory of GSE research methods underscores a meaningful alignment between research and industry. This implies that researchers are proactive in addressing the evolving challenges stemming from developing sophisticated games. This alignment fosters a symbiotic relationship between academia and

industry, as research innovations are poised to directly contribute to improving game development processes.

1.5.4 RQ4 Empirical Methods

The fact that case studies are the empirical method that has been most widely conducted by researchers since 2009. This can be a sign that the gap between industry and academic research is smaller since surveys were previously the majority, and now case studies are. In the past, companies were only involved in research at a surface level, answering surveys. Now there are plenty of case studies that nourish GSE.

Experiments were also expected to increase since 2009 [5], but the research evidence does not support this hypothesis. The video game industry is fast-paced, having less development time than traditional software [56]. This rush can explain the reduced number of experiments applied. Experiments require more effort and are more time-consuming than case studies, which have a strong short-term impact with direct implementation of the product.

GSE research makes use of in-depth case studies that are drawn from industrial game development initiatives. This collaborative synergy not only strengthens the relevance of GSE research but also leads to the creation of practical tools and techniques that address the specific needs of the gaming industry. As companies openly share their experiences and challenges, the research community gains access to invaluable data and context, ultimately reducing the gap between theoretical advancements and pragmatic modern game development.

1.5.5 RQ5 Research Quality

In Section 1.4.5, we stated that best practices or widely accepted methods to report limitations are only used by seven studies. This situation can be explained by the lack of exploration in GSE research strategies and practices due to the empirical nature and relative youth of this field.

The average score of all the studies is 3.49, with a standard deviation of 1.34. Although the field of research has matured to the point where there

are a lot of industry-scale demonstrations, the study reporting needs to be more rigorous.

It becomes imperative to ensure that research outcomes are communicated with a higher degree of clarity, transparency, and repeatability. Researchers must acknowledge the importance of adopting standardized methodologies, rigorous experimental designs, and systematic documentation practices to effectively validate and communicate their findings. This will lead to meaningful and informed decision-making in both academia and industry.

Key insights

- RQ1:** The landscape of industry-scale GSE research is witnessing an x4 growth, with 98 studies published across 86 distinct venues. This surge in activity underscores the intensifying interest in GSE, a field that is evolving but is dispersed.
- RQ2:** The shift in the topic interest by the researchers indicates an increase in the maturity of industry-scale GSE research. As the video game market becomes more competitive, GSE research provides more sophisticated techniques and methods.
- RQ3:** Recent years have seen a remarkable doubling in interest in empirical methods, reflecting the maturation of GSE. Industry recognition of collaboration benefits has led to an active increase in empirical studies, narrowing the industry-academia gap.
- RQ4:** Case studies have surged as the primary empirical method used by researchers since 2009. This shift suggests a further reduction in the industry-academia gap, marking a transition from surveys to more immersive case studies. This change reflects deeper industry involvement in GSE, showcasing a richer understanding of real-world scenarios.
- RQ5:** While the research field has progressed, marked by a substantial presence of industry-scale demonstrations, there is still a demand for increased rigor in study reporting.

1.6 Future Directions

In this section, we analyze the different research challenges that are open in the field of work of GSE. We present in the following subsections the challenges, why they exist, and the open problems for each challenge. We also provide recommendations and potential future research lines in the identified areas.

1.6.1 *Cross-fertilization between GSE and MDE*

GSE is a fertile field for Model Driven Engineering (MDE) research to thrive [S18, S22, S19, S36, S35]. The heterogeneity of the profiles involved in the creation of a video game makes it necessary to have approaches that allow profiles without requiring specific programming knowledge to participate in the creation process. In engines such as *Unity* and *Unreal Engine*, where many development processes have been presented in an abstract way through models, templates, blueprints, and flowcharts, reducing the prominence of the source code generated from these models. This practice is what the MDE research community has been pursuing for decades in classic software engineering. Nevertheless, the MDE research community does not seem to be fully aware of the modeling success achieved by video games given the tiny presence of GSE work in MDE venues. There is an opportunity for the MDE research community to learn from the success of models in video games in the hope of transferring it to other SE domains. It also provides the GSE community with an opportunity to leverage MDE approaches for video game models.

1.6.2 *Game Software Product Lines*

GSE is a domain where software reuse is constant, either adapting systems and elements from different games in a company or within the same game (the same element can appear in different parts of the same game) [S98, S89, S36, S20, S92, S8]. The urgency of the sector to meet deadlines hinders the investment of time in developing and implementing specific techniques for code reuse. Some advances have been made towards this, such as the development of game engines [S95, S50]. However, other advances in reuse have not yet been applied with the same

success as in the case of software product lines and the planned reuse approach they propose. We think that in the near future there will be a growing body of work from the GSE reuse community that leverages SE reuse techniques to the GSE world and adapts them for maximum benefit. Software reuse is also a key feature in porting video games to different platforms.

1.6.3 The Blossoming of Publishing Platforms

There are many platforms on which to publish video games. With the growth of the video game industry, multi-platform publishing is the norm. These platforms have different input schemes, different computing power, different audiences, different operating systems, and different hardware [S98, S92, S41, S1]. Video games need to comply with the high industry standards in all of the platforms on which they release their software. These platforms also change rapidly with new console and hardware releases. Porting a game from one platform to another is not a trivial task and developers spend a lot of resources doing it. However, GSE research neglects this topic. We expect more work on GSE techniques and tools that help developers when porting video games across platforms.

1.6.4 The Multifaceted Nature of GSE

As previously mentioned in Section 1.5.2, one challenge that most research in software engineering has (traditional or video games) is the need for generalization of findings. The approach reported in GSE work is usually evaluated in a specific environment that cannot be generalized directly to all cases [S55, S54, S34, S62, S2, S13]. In video games, this problem is amplified due to the inherent heterogeneity of the field. Video games are being made by teams of very different sizes, from solo indie developers to big corporations with hundreds of employees. There are many platforms with plenty of differences among them. There are also plenty of genres that change the software significantly, from physics-based action games in 3D to puzzle games in 2D. The underlying software that composes the genres amplifies the differences in the software between games (and sometimes in the same game), making it difficult to general-

ize findings for GSE research. There is a need to tackle the challenge of generalization in the context of GSE settings.

1.6.5 Planning and Scheduling Issues

The video game industry focuses on agile methodologies. Many report that agile is not properly implemented in the teams and there are misconceptions about different agile methodologies [S66, S70, S71, S77]. Crunch time is a term used in the game industry to describe high-stress periods working overtime that can last months when releasing a video game [S75]. The multidisciplinary teams in game development are using agile methodologies as a legacy from traditional software engineering [7], but these methods suffered from crunch time [S75]. The way in which many companies are adopting existing SE methodologies is clearly not the optimal solution when developers need to rely on crunch time. The GSE community can put effort into finding new best practices for creating games and adapting agile methods for game development teams of different sizes in order to avoid crunch culture.

1.6.6 The Consequences of Games as a Service

The GSE community is expected to follow the growth, maturity, and needs of the industry. The five topics that show the greatest interest from practitioners (D.2.2 Design Tools and Techniques, D.2.5 Testing and Debugging, D.2.8 Metrics, D.2.9 Management, and D.2.11 Software Architectures) will probably hold this research intensity over the next few years as they tackle the problems in an industry that keeps growing in projects and team sizes. Nevertheless, a part of the video game industry has veered towards a business model approach called Games as a Service (GaaS) [57]. Similar to the concept of Software as a Service (SaaS), these are products that do not end development upon release. For this reason, topic D.2.7 Distribution, Maintenance, and Enhancement may see an increase in engagement by both industry and the research community. Because of the GaaS trend, software maintenance will be a major issue in the video game industry in the future [S54, S55, S62, S2, S94, S93].

1.6.7 GSE Security

Multiplayer online games comprise a significant part of the user base of video games. These games need to build network systems that are reliable, robust, and secure [S93, S94, S83, S2]. Online video games synchronize large amounts of data provided by different platforms without latency in order to be competitive. With the rise of Massive Multiplayer Online (MMO) games, bandwidth usage and high traffic are issues that have greater complexity in the GSE domain. These games portray large complex 3D worlds with physics calculations involved and many players interacting with each other. Online games can also be competitive in nature, giving users a platform for what is called e-sports (tournaments where professional video game players compete). This raises the concern of cheaters and the need for anti-cheat technology that detects players who exploit the vulnerabilities of the software.

1.6.8 Weaving Software and Art Design

The fact that software, art, design, and sound are tightly connected in video games poses difficulties in documenting the development. Practitioners cannot produce software without taking into account the design that will make the game fun and the art and sound that will surround the experience. The communication among the multidisciplinary team members must be clear and fluid. The GSE research community is already putting effort into this topic by proposing improvements for these documents called Game Design Documents (GDDs) [S4, S5, S6, S78, S79, S80, S77]. Nonetheless, there is no established standard for the making of GDDs; every practitioner develops these documents in their own fashion. To address the evolving needs of game development teams, future directions must focus on improving and exploring methods, possibly veering out of the traditional GDD creation due to the static nature of GDDs that hinders iteration and prototyping [58].

1.6.9 A New Wave of Requirements Research for GSE

Video game development focuses on the fun and entertainment of the players. Traditional SE usually does not bother with this, making GSE a domain that needs a change in how researchers approach requirements engineering (RE). Even though RE is a topic that has lost attention from the GSE community, research still needs to be done in this field. RE can help define the final user experience desired by developers. It can facilitate tasks of gameplay testing because quality assurance developers will have a clearer objective when testing. We think that, in the future, there will be more RE work applied to GSE that properly exploits the particularities of the domain (e.g., in fields such as user experience evaluation or video game balancing) [S7, S3, S4, S5, S6].

1.6.10 Testing and Debugging in the Face of Gameplay Subjectivity

Similarly to RE, testing in video games is peculiar due to the player's subjectivity for fun. This divides testing in two when applied to GSE: traditional testing, which searches for software bugs and faults that make the program misbehave [S38, S39, S40, S41, S46, S48]; and gameplay testing, which affects how the user perceives the experience (e.g., the game is unbalanced and unfair) [S42, S43, S44, S49]. GSE has to deal with large code bases with high interactivity among objects and large amounts of assets that make bug reproduction a difficult task that needs to be addressed. Additionally, when performing gameplay testing, the human tester has to spot bugs while playing the game, making this process subjective to human opinion and ideas. There is a need for automation and new techniques in the testing and debugging domain.

1.6.11 Green GSE

Video games need to comply with high industry standards in terms of performance while creating complex 3D worlds. GSE is a field that needs to take advantage of the hardware acceleration of graphic cards to provide stable execution of the game with constant Frames Per Second (FPS). This makes code performance a significant concern to developers [S86, S89]. It can also affect other aspects of the game and the user experience

such as long loading screens, which are states in which the game loads all the needed assets. Additionally, low performance in code leads to high energy usage that affects portable machines such as laptops and mobile phones. Reducing energy usage with better performant software also plays an important role in ecology [59] since millions of players are consuming video games. In the end, having fast code in video games is an open issue that has many implications and the next generation of games will need improvements in code performance.

1.6.12 Search-Based GSE

Search-Based Software Engineering (SBSE) is a steady and popular topic among the SE community [60]. The key ingredient for SBSE approaches is the fitness function that guides the search. In video games, there are Non-Playable Characters (NPCs), agents with artificial intelligence that can behave like the player. These NPCs can be used to perform simulations, which are an opportunity to guide the search in SBSE, improving the fitness function. Also, the high user base of video games (blockbusters gather hundreds of millions of players) can contribute to the vision of Humans as Fitness Function [61], where humans share the load of assessing candidate solutions.

1.6.13 Content Creation and Artificial Intelligence

The scope of this review has omitted content creation and artificial intelligence. There is also a need to analyze the state of the art in these domains and how GSE is present in them.

1.7 Threats to Validity and Quality Assessment

1.7.1 Threats to validity

Wohlin et al. propose a classification for assessing the validity of the results with four categories of validity concerns [62, 63]. In this section, we test our review against these categories.

Internal Validity: This category can be a validity threat when the results of an investigation may be affected by factors affecting the dependent variables without the researcher's knowledge. In our work, this could be a threat when we selected the studies to review. To reduce the likelihood of erroneous exclusions or inclusions, we defined the criteria for when a study can be excluded or included in Section 1.3.2. We used the same criteria for evaluating all of the studies from the total studies list to the final selection. In the same way, we performed the data collection and the quality assessment of each selected study following the criteria defined in Sections 1.3.4 and 1.3.3, respectively.

External Validity: This category addresses the likelihood of generalizing the results of the findings. In order to minimize the threat of the primary studies collected from the databases not being representative of the target population, we used the ACM and IEEE databases in addition to two indexing databases (Scopus and Web of Science), in the way suggested by Petersen et al. [46]. We reported this process in Section 1.3.2.

Conclusion Validity: This category is concerned with the reliability of the relationship between the treatments and the outcome of an experiment. In our case, the results might be compromised if researchers were looking for a specific outcome. To avoid this threat, we conducted the search method using best practices in SLRs [34, 45, 64]. The only case for discarding a study was if it did not match the inclusion criteria or if it matched any exclusion criteria. Also, as explained in Section 1.3.2, two reviewers performed the selection process, and all disagreements were resolved by discussing the decision of each reviewer and consulting a third author if necessary.

Construct Validity: This category is related to the concepts, theories, and measures that the researchers have in mind. *Examples of issues are whether the concepts are defined clearly enough before measurements are defined, and interaction of different treatments when persons are involved in more than one study* [62]. To prevent this threat, we defined the research questions for researchers to know what to investigate, and we defined a search process for all of the

reviewers of this work to follow. The research questions are provided in Section 1.3.1, and the search process is explained in Section 1.3.2.

1.7.2 Assessment of review

As an assessment of this review, we have performed an evaluation using four quality assessment (QA) questions from the work of Kitchenham et al. [36] about systematic literature reviews in software engineering. We answer the questions by using the specified criteria by Kitchenham et al.

QA1: *Are the review's inclusion and exclusion criteria described and appropriate?*

Yes, we explicitly defined the inclusion criteria in the study.

QA2: *Is the literature search likely to have covered all relevant studies?*

Yes, the authors have searched four digital libraries with the extra search strategy of snowballing.

QA3: *Did the reviewers assess the quality/validity of the included studies?*

Yes, the authors have explicitly defined quality criteria and extracted them from each primary study.

QA4: *Were the basic data/studies adequately described?*

Yes, we presented information about each study.

These quality assessment criteria might be biased as they are somewhat subjective. In order to minimize erroneous results, two researchers evaluated these quality criteria independently. A different researcher outside of the review process resolved all discrepancies.

1.7.3 Other threats and limitations

Another threat to the validity of our study is due to the fact that 8.08% of our final selection are workshop reports. The inclusion of these studies in the SLR might have altered the review results due to the nature of these studies with respect to journals and conferences.

Finally, it is worth noting that our review only focuses on computer games and industry-scale studies. Therefore, it might not be safe to generalize it to the whole GSE research. These would also include studies focusing on non-computer games and studies with evidence obtained from academic studies, expert opinions or observations, demonstration or working out academic prototypes, or with no evidence at all.

1.8 Conclusion

This study presents an SLR to study software engineering research for industry-scale computer games. This review comprises 98 studies portraying the current relationship between GSE academia and industry.

Our findings reveal that research on GSE continues increasing in interest and that the industry benefits from it (with over four times more research than before 2009). Nevertheless, these studies need to be more rigorous with the critical examination of their work.

We have observed an evolution in the research topics that draw the most attention from practitioners. The categories related to a young domain are decreasing, leading to a more mature field with research about architecture and design techniques as the main topics. We expect this trend to continue, with topics related to software maintenance and games as a service (GaaS) gaining relevance in the upcoming years.

The 2012 ACM CCS does not correctly represent the GSE research domain. The studies that we reviewed were unevenly distributed in this categorization. This contrast supports the claim that GSE and traditional SE are diverging. A new classification for GSE might be needed in the near future. In addition, the GSE research community is completely dispersed, with no main venues holding GSE studies yet. We also expect

specialized journals and conferences on GSE research to start appearing over the next few years.

The research approaches and empirical methods used by the community support the hypothesis that GSE is an independent, mature, and growing domain. Our study shows how the gap between industry and scientific research is narrowing, even in the rapidly changing industry of video games.

Our results confirm the assumptions from the previous work by Ampatzoglou and Stamelos [5]: *“more elaborate empirical research methods are going to be employed”, “a wider range of topics is going to be covered”, and “game engineering is a scientific domain rather than a soft skill topic with only a peripheral research activity”.*

This study provides background information for any relevant future GSE work, helping the community to explore the research gaps that GSE presents within an industry that shows great interest in academic research. This indicates a bright future for the growing field of Game Software Engineering. While GSE is still a young field, it is growing and maturing each day. We presented many lines of research for the future, making GSE a fertile domain that is constantly growing.

According to the report provided by S. Shuermans and C. Voskoglou in 2019 [65], there are 18.9M software developers, of which 8.8M are game developers. The research in GSE is relevant to almost half of the software developers in the world and is, directly or indirectly, relevant to all the roles that deal with the other aspects of development: art, music, design, narrative, etc.

The large democratization of game development poses an opportunity for the mainstream engines (Unity and Unreal) to teach and divulge software engineering practices for developers that do not have an engineering background. Developers that built expertise in these engines often lack experience in SE methods and could benefit from them. Not only can large organizations benefit from GSE research, but small video game studios can also cover many roles with a reduced team. In these scenarios, automatizing processes, performing accessible testing practices, or team

management methodologies can promote serious changes in development. They can facilitate the technical process and reduce the technical debt of developers so that they can focus on the creative and fun part of the video game craft.

This study provides a bridge between industry and academia so that they can nourish each other. It shows traditional SE researchers that GSE deals with pieces of software that are complex enough to address most topics that traditional SE also covers. Researchers can find complete and industrial cases for their studies inside video game development. By providing a large sample of research studies in GSE, we hope that industrial practitioners are encouraged to publish their industrial-scale expertise. Video games are powerful tools that can push the boundaries of software knowledge.

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 S05_20D).

Studies Included in the Review

- [S1] Lei Chen et al. “Investigating the security and digital forensics of video games and gaming systems: A study of PC games and PS4 console”. In: *2016 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2016, pp. 1–5 (cit. on pp. 49, 60, 65).
- [S2] Blake D Bryant and Hossein Saiedian. “A State Saturation Attack against Massively Multiplayer Online Videogames.” In: *ICISSP (2021)*, pp. 217–225 (cit. on pp. 49, 60, 65–67).

- [S3] Jassin Kessing, Tim Tutenel, and Rafael Bidarra. “Designing semantic game worlds”. In: *Proceedings of The third workshop on Procedural Content Generation in Games*. 2012, pp. 1–9 (cit. on pp. 49, 60, 68).
- [S4] Carlos Marín-Lora, Miguel Chover, and Jose M Sotoca. “A Game Logic Specification Proposal for 2D Video Games”. In: *World Conference on Information Systems and Technologies*. Springer. 2020, pp. 494–504 (cit. on pp. 49, 60, 67, 68).
- [S5] Carlos Marín-Lora et al. “A First Step to Specify Arcade Games as Multi-agent Systems”. In: *World Conference on Information Systems and Technologies*. Springer. 2020, pp. 369–379 (cit. on pp. 49, 60, 67, 68).
- [S6] Carlos Marín-Lora, Miguel Chover, and Jose M Sotoca. “A Multi-agent Specification for the Tetris Game”. In: *International Symposium on Distributed Computing and Artificial Intelligence*. Springer. 2021, pp. 169–178 (cit. on pp. 49, 60, 67, 68).
- [S7] Jussi Kasurinen, Andrey Maglyas, and Kari Smolander. “Is requirements engineering useless in game development?” In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2014, pp. 1–16 (cit. on pp. 49, 60, 68).
- [S8] Jussi Kasurinen, Jukka-Pekka Strandén, and Kari Smolander. “What do game developers expect from development and design tools?” In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*. 2013, pp. 36–41 (cit. on pp. 49, 60, 64).
- [S9] Jozef Kulik, Jen Beeston, and Paul Cairns. “Grounded theory of accessible game development”. In: *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*. 2021, pp. 1–9 (cit. on p. 49).
- [S10] Maya Daneva. “How practitioners approach gameplay requirements? An exploration into the context of massive multiplayer online role-playing games”. In: *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE. 2014, pp. 3–12 (cit. on p. 49).

- [S11] José L González Sánchez, Rosa M Gil Iranzo, and Francisco L Gutiérrez Vela. “Enriching evaluation in video games”. In: *IFIP Conference on Human-Computer Interaction*. Springer. 2011, pp. 519–522 (cit. on p. 49).
- [S12] José Luis González Sánchez et al. “Playability: analysing user experience in video games”. In: *Behaviour & Information Technology* 31.10 (2012), pp. 1033–1054 (cit. on p. 49).
- [S13] Stephen Vickers, Howell Istance, and Matthew Smalley. “EyeGuitar: making rhythm based music video games accessible using only eye movements”. In: *Proceedings of the 7th international conference on advances in computer entertainment technology*. 2010, pp. 36–39 (cit. on pp. 49, 60, 65).
- [S14] Howell Istance et al. “Designing gaze gestures for gaming: an investigation of performance”. In: *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*. 2010, pp. 323–330 (cit. on p. 49).
- [S15] Nafa Zulfa et al. “The Effect of UI and UX Enhancement on Bomberman Game Based on Game Experience Questionnaire (GEQ)”. In: *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*. IEEE. 2020, pp. 543–547 (cit. on p. 49).
- [S16] Kristine Jørgensen. “Between the game system and the fictional world: a study of computer game interfaces”. In: *Games and Culture* 7.2 (2012), pp. 142–163 (cit. on p. 49).
- [S17] Joe Nandhakumar, Nikiforos S Panourgias, and Harry Scarbrough. “From knowing it to “getting it”: Envisioning practices in computer games development”. In: *Information Systems Research* 24.4 (2013), pp. 933–955 (cit. on p. 49).
- [S18] Silvia Abrahão et al. “Early usability in model-driven game development”. In: *International Conference on Product-Focused Software Process Improvement*. Springer. 2016, pp. 713–722 (cit. on pp. 49, 60, 64).
- [S19] África Domingo et al. “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*. Ed. by Schahram Dustdar et al.

- Vol. 12127. Lecture Notes in Computer Science. Springer, 2020, pp. 353–367 (cit. on pp. 49, 60, 64).
- [S20] Ely Fernando do Prado and Daniel Lucrédio. “A flexible model-driven game development approach”. In: *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software*. IEEE. 2015, pp. 130–139 (cit. on pp. 49, 60, 64).
- [S21] Guilherme Willian De Oliveira, Stéphane Julia, and Ligia Maria Soares Passos. “Game modeling using WorkFlow nets”. In: *2011 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE. 2011, pp. 838–843 (cit. on p. 49).
- [S22] Franciny M Barreto and Stéphane Julia. “Modeling and analysis of video games based on workflow nets and state graphs”. In: *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. 2014, pp. 106–119 (cit. on pp. 49, 60, 64).
- [S23] Franciny M Barreto and Stéphane JULIA. “Formal Approach Based on Petri Nets for Modeling and Verification of Video Games”. In: *Computing and Informatics* 40.1 (2021), pp. 216–248 (cit. on p. 49).
- [S24] África Domingo et al. “Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives”. In: *Advanced Information Systems Engineering*. Ed. by Marcello La Rosa, Shazia Sadiq, and Ernest Teniente. Cham: Springer International Publishing, 2021, pp. 483–498. ISBN: 978-3-030-79382-1 (cit. on p. 49).
- [S25] Henrik Engström and Per Anders Östblad. “Using text-to-speech to prototype game dialog”. In: *Computers in Entertainment (CIE)* 16.4 (2018), pp. 1–16 (cit. on p. 50).
- [S26] Micah J Best et al. “Collection-focused parallelism”. In: *5th USENIX Workshop on Hot Topics in Parallelism (HotPar 13)*. 2013 (cit. on p. 50).
- [S27] Halimatus Sa’dyah et al. “The fundamental topics of static global illumination algorithms for 3d games”. In: *2018 IEEE 3rd international conference on communication and information systems (ICCIS)*. IEEE. 2018, pp. 237–241 (cit. on p. 50).

- [S28] Huang Xu and Cui Ping Wang. “A Review and Development of 3-D Accelerator Technology for Games”. In: *2009 Second International Symposium on Intelligent Information Technology and Security Informatics*. IEEE. 2009, pp. 59–63 (cit. on p. 50).
- [S29] Yong He et al. “Shader components: modular and high performance shader development”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–11 (cit. on p. 50).
- [S30] Vladeta Stojanovic et al. “Comparison of advanced and standard real-time 3D rendering methods for interactive landscapes (Short paper version)”. In: *2013 17th International Conference on Information Visualisation*. IEEE. 2013, pp. 539–544 (cit. on p. 50).
- [S31] Kerry A Seitz Jr et al. “Staged metaprogramming for shader system development”. In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–15 (cit. on p. 50).
- [S32] Aznam Yacoub, Maamar El Amine Hamri, and Claudia Frydman. “DEV-PROMELA: modeling, verification, and validation of a video game by combining model-checking and simulation”. In: *Simulation* 96.11 (2020), pp. 881–910 (cit. on p. 50).
- [S33] Ruslan Rezin et al. “Model checking in multiplayer games development”. In: *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE. 2018, pp. 826–833 (cit. on p. 50).
- [S34] Sha Ding et al. “RTS-GameFlow: a new evaluation framework for RTS games”. In: *2009 International Conference on Computational Intelligence and Software Engineering*. IEEE. 2009, pp. 1–4 (cit. on pp. 50, 60, 65).
- [S35] Adrian Fernandez et al. “Integrating usability evaluation into model-driven video game development”. In: *International Conference on Human-Centred Software Engineering*. Springer. 2012, pp. 307–314 (cit. on pp. 50, 60, 64).
- [S36] Nao Igawa et al. “Model Checking of Visual Scripts Created by UE4 Blueprints”. In: *2020 9th International Congress on Advanced Applied Informatics (IIAI-AAI)*. IEEE. 2020, pp. 512–515 (cit. on pp. 50, 60, 64).

- [S37] Shuji Morisaki et al. “Detecting source code hotspot in games software using call flow analysis”. In: *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE. 2019, pp. 484–489 (cit. on p. 50).
- [S38] Mohammad Ghoreschi and Hassan Haghighi. “An incremental method for extracting tests from object-oriented specification”. In: *Information and Software Technology* 78 (2016), pp. 1–26 (cit. on pp. 50, 60, 68).
- [S39] Bhargava Rohit Sagi and Rachel Silvestrini. “Application of combinatorial tests in video game testing”. In: *Quality Engineering* 29.4 (2017), pp. 745–759 (cit. on pp. 50, 60, 68).
- [S40] Isamu Hasegawa and Tomoyuki Yokogawa. “Automatic verification for node-based visual script notation using model checking”. In: *International Conference on Formal Engineering Methods*. Springer. 2019, pp. 52–68 (cit. on pp. 50, 60, 68).
- [S41] Sidra Iftikhar et al. “An automated model based testing approach for platform games”. In: *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2015, pp. 426–435 (cit. on pp. 50, 60, 65, 68).
- [S42] Pejman Mirza-Babaei, Naeem Moosajee, and Brandon Drenikow. “Playtesting for indie studios”. In: *Proceedings of the 20th International Academic Mindtrek Conference*. 2016, pp. 366–374 (cit. on pp. 51, 60, 68).
- [S43] Pejman Mirza-Babaei et al. “A Postmortem on Playtesting: Exploring the Impact of Playtesting on the Critical Reception of Video Games”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–12 (cit. on pp. 51, 60, 68).
- [S44] Xinge Tong. “Positioning game review as a crucial element of game user feedback in the ongoing development of independent video games”. In: *Computers in Human Behavior Reports* 3 (2021), p. 100077 (cit. on pp. 51, 60, 68).

- [S45] Drew Zagieboylo and Kazi A Zaman. “Cost-Efficient and Reliable Reporting of Highly Bursty Video Game Crash Data”. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. 2017, pp. 201–212 (cit. on p. 51).
- [S46] Dario Festa et al. “Supporting distributed real-time debugging in online games”. In: *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2017, pp. 737–740 (cit. on pp. 51, 60, 68).
- [S47] Andrew Truelove, Eduardo Santana de Almeida, and Iftekhhar Ahmed. “We’ll fix it in post: what do bug fixes in video game update notes tell us?” In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 736–747 (cit. on p. 51).
- [S48] Chris Lewis and Jim Whitehead. “Repairing games at runtime or, how we learned to stop worrying and love emergence”. In: *IEEE software* 28.5 (2011), pp. 53–59 (cit. on pp. 51, 60, 68).
- [S49] Jussi Kasurinen and Kari Smolander. “What do game developers test in their products?” In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2014, pp. 1–10 (cit. on pp. 51, 60, 68).
- [S50] Samuel Lynch et al. “M22-A Modern Visual Novel Framework”. In: *Proceedings of the 8th International Workshop on Narrative and Hypertext*. 2019, pp. 9–13 (cit. on pp. 51, 60, 64).
- [S51] Cristiano Politowski et al. “Are game engines software frameworks? a three-perspective study”. In: *Journal of Systems and Software* 171 (2021), p. 110846 (cit. on p. 51).
- [S52] Daniel MacCormick and Loutfouz Zaman. “SuBViS: the use of subjunctive visual programming environments for exploring alternatives in game development”. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 2019, pp. 1–11 (cit. on p. 51).

- [S53] Timo Nummenmaa et al. “Applying lehman’s laws to game evolution”. In: *Proceedings of the 2013 International Workshop on Principles of Software Evolution*. 2013, pp. 11–17 (cit. on p. 52).
- [S54] Artian Kica et al. “Nerfs, buffs and bugs-analysis of the impact of patching on League of Legends”. In: *2016 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE. 2016, pp. 128–135 (cit. on pp. 52, 60, 65, 66).
- [S55] Xiaofang Zhong and Jinjie Xu. “Game updates enhance players’ engagement: a case of DOTA2”. In: *2021 4th International Conference on Information Management and Management Science*. 2021, pp. 117–123 (cit. on pp. 52, 60, 65, 66).
- [S56] Anders Drachen and Alessandro Canossa. “Analyzing spatial user behavior in computer games using geographic information systems”. In: *Proceedings of the 13th international MindTrek conference: Everyday life in the ubiquitous era*. 2009, pp. 182–189 (cit. on p. 52).
- [S57] Troy Costa Kohwalter, Leonardo Gresta Paulino Murta, and Esteban Walter Gonzalez Clua. “Capturing game telemetry with provenance”. In: *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE. 2017, pp. 66–75 (cit. on p. 52).
- [S58] Kenneth Hullett et al. “Data analytics for game development: NIER track”. In: *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE. 2011, pp. 940–943 (cit. on p. 52).
- [S59] Lidson Barbosa Jacob et al. “A non-intrusive approach for 2d platform game design analysis based on provenance data extracted from game streaming”. In: *2014 Brazilian Symposium on Computer Games and Digital Entertainment*. IEEE. 2014, pp. 41–50 (cit. on p. 52).
- [S60] Yegeta Zeleke, Joseph C Osborn, and Ricardo G Sanfelice. “Analyzing action games: a hybrid systems approach”. In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 2019, pp. 1–11 (cit. on p. 52).

- [S61] Kenneth Hullett et al. “Empirical analysis of user data in game software development”. In: *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE. 2012, pp. 89–98 (cit. on p. 52).
- [S62] John L Miller and Jon Crowcroft. “Avatar movement in World of Warcraft battlegrounds”. In: *2009 8th Annual Workshop on Network and Systems Support for Games (NetGames)*. IEEE. 2009, pp. 1–6 (cit. on pp. 52, 60, 65, 66).
- [S63] Spyridon Papaloukas, Kiriakos Patriarcheas, and Michalis Xenos. “Usability assessment heuristics in new genre videogames”. In: *2009 13th Panhellenic Conference on Informatics*. IEEE. 2009, pp. 202–206 (cit. on p. 52).
- [S64] Penelope Sweetser et al. “GameFlow heuristics for designing and evaluating real-time strategy games”. In: *Proceedings of the 8th Australasian Conference on Interactive Entertainment: Playing the System*. 2012, pp. 1–10 (cit. on p. 52).
- [S65] Rosa Yanez-Gomez et al. “Heuristic usability evaluation on games: a modular approach”. In: *Multimedia Tools and Applications* 78.4 (2019), pp. 4937–4964 (cit. on p. 52).
- [S66] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed. “A digital game maturity model (DGMM)”. In: *Entertainment Computing* 17 (2016), pp. 55–73 (cit. on pp. 52, 60, 66).
- [S67] Jussi Kasurinen and Kari Smolander. “Defining an iterative ISO/IEC 29110 deployment package for game developers”. In: *International Journal of Information Technologies and Systems Approach (IJITSA)* 10.1 (2017), pp. 107–125 (cit. on p. 52).
- [S68] Christopher M Kanode and Hisham M Haddad. “Software engineering challenges in game development”. In: *2009 Sixth International Conference on Information Technology: New Generations*. IEEE. 2009, pp. 260–265 (cit. on p. 52).

- [S69] Alia Fatima, Tayyaba Rasool, and Usman Qamar. “GDGSE: Game Development with Global Software Engineering”. In: *2018 IEEE Games, Entertainment, Media Conference (GEM)*. IEEE. 2018, pp. 1–9 (cit. on p. 52).
- [S70] Harry Scarbrough, Niki Panourgias, and Joe Nandhakumar. “The role of objects in the coordination of knowledge-intensive projects: A study of computer games development”. In: *2012 45th Hawaii International Conference on System Sciences*. IEEE. 2012, pp. 4952–4960 (cit. on pp. 52, 60, 66).
- [S71] Rido Ramadan and Yani Widayani. “Game development life cycle guidelines”. In: *2013 International Conference on Advanced Computer Science and Information Systems (ICACISIS)*. IEEE. 2013, pp. 95–100 (cit. on pp. 52, 60, 66).
- [S72] Norita B Ahmad et al. “How to launch a successful video game: A framework”. In: *Entertainment computing 23* (2017), pp. 1–11 (cit. on p. 52).
- [S73] Rudy McDaniel. “Communication and knowledge management strategies in video game design and development: A case study highlighting key organizational narratives”. In: *2015 IEEE International Professional Communication Conference (IPCC)*. IEEE. 2015, pp. 1–8 (cit. on p. 52).
- [S74] Marc Schmalz, Aimee Finn, and Hazel Taylor. “Risk management in video game development projects”. In: *2014 47th Hawaii International Conference on System Sciences*. IEEE. 2014, pp. 4325–4334 (cit. on p. 52).
- [S75] Henrik Edholm et al. “Crunch time: The reasons and effects of unpaid overtime in the games industry”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE. 2017, pp. 43–52 (cit. on pp. 52, 60, 66).
- [S76] Klara Borowa, Andrzej Zalewski, and Adam Saczko. “Living With Technical Debt—A Perspective From the Video Game Industry”. In: *IEEE Software* 38.06 (2021), pp. 65–70 (cit. on p. 52).
- [S77] Jing-Wei Liu et al. “The role of Sprint planning and feedback in game development projects: Implications for game quality”. In: *Journal of Systems and Software* 154 (2019), pp. 79–91 (cit. on pp. 53, 60, 66, 67).

- [S78] Megan A Winget and Wiliam Walker Sampson. “Game development documentation and institutional collection development policy”. In: *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*. 2011, pp. 29–38 (cit. on pp. 53, 60, 67).
- [S79] Mario Gonzalez Salazar et al. “Proposal of Game Design Document from software engineering requirements perspective”. In: *2012 17th International Conference on Computer Games (CGAMES)*. IEEE. 2012, pp. 81–85 (cit. on pp. 53, 60, 67).
- [S80] Emmanuel Guardiola. “The gameplay loop: a player activity model for game design and analysis”. In: *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*. 2016, pp. 1–7 (cit. on pp. 53, 60, 67).
- [S81] Magy Seif El-Nasr. “Developing games that capture and engage users”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE. 2019, pp. 9–10 (cit. on p. 53).
- [S82] Walt Scacchi. “Modding as an open source approach to extending computer game systems”. In: *IFIP International Conference on Open Source Systems*. Springer. 2011, pp. 62–74 (cit. on p. 53).
- [S83] Mirko Suznjevic, Ivana Stupar, and Maja Matijasevic. “A model and software architecture for MMORPG traffic generation based on player behavior”. In: *Multimedia systems* 19.3 (2013), pp. 231–253 (cit. on pp. 53, 60, 67).
- [S84] Blake Bryant and Hossein Saiedian. “An evaluation of videogame network architecture performance and security”. In: *Computer Networks* 192 (2021), p. 108128 (cit. on p. 53).
- [S85] Michael Kenzel et al. “A high-performance software graphics pipeline architecture for the GPU”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–15 (cit. on p. 53).

- [S86] Alf Inge Wang and Njål Nordmark. “Software architectures and the creative processes in game development”. In: *International Conference on Entertainment Computing*. Springer. 2015, pp. 272–285 (cit. on pp. 53, 68).
- [S87] Tobias Olsson et al. “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 (cit. on p. 53).
- [S88] Tobias Ollsson et al. “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 (cit. on p. 53).
- [S89] Ali Mohebbi and Thiam Kian Chiew. “Redefining game engine architecture through concurrency”. In: *International Conference on Intelligent Software Methodologies, Tools, and Techniques*. Springer. 2014, pp. 149–161 (cit. on pp. 53, 60, 64, 68).
- [S90] Wilson Kazuo Mizutani and Fabio 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 (cit. on p. 53).
- [S91] Timo Nummenmaa, Eleni Berki, and Tommi Mikkonen. “Exploring games as formal models”. In: *2009 Fourth South-East European Workshop on Formal Methods*. IEEE. 2009, pp. 60–65 (cit. on p. 53).
- [S92] Arati Srinivasan and Venkat N Venkatraman. “Architectural convergence and platform evolution: Empirical test of complementor moves in videogames”. In: *IEEE Transactions on Engineering Management* 67.2 (2018), pp. 266–282 (cit. on pp. 53, 60, 64, 65).
- [S93] J. Kienzle and A. Denault. “Journey: A Massively Multiplayer Online Game Middleware”. In: *IEEE Software* 28.05 (2011), pp. 38–44. ISSN: 1937-4194 (cit. on pp. 54, 60, 66, 67).
- [S94] Jesse Donkervliet, Jim Cuijpers, and Alexandru Iosup. “Dyconits: Scaling Minecraft-like Services through Dynamically Managed Inconsistency”.

- In: *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2021, pp. 126–137 (cit. on pp. 54, 60, 66, 67).
- [S95] Filipe Boaventura and Victor T Sarinho. “Mendiga: A minimal engine for digital games”. In: *International Journal of Computer Games Technology* 2017 (2017) (cit. on pp. 54, 60, 64).
- [S96] Benjamin J. Geisler, Frank J. Mitropoulos, and Shane Kavage. “GAME-SPECT: Aspect Oriented Programming for a Video Game Engine using Meta-languages”. In: *2019 SoutheastCon*. 2019, pp. 1–8 (cit. on p. 54).
- [S97] Ben J. Geisler and Shane L. Kavage. “A Multi-engine Aspect-Oriented Language with Modeling Integration for Video Game Design”. In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by Raian Ali, Hermann Kaindl, and Leszek A. Maciaszek. Cham: Springer International Publishing, 2021, pp. 336–359. ISBN: 978-3-030-70006-5 (cit. on p. 54).
- [S98] Fabrizio Nunnari and Alexis Héloir. “Write-Once, Transpile-Everywhere: Re-using Motion Controllers of Virtual Humans Across Multiple Game Engines”. In: July 2018, pp. 435–446. ISBN: 978-3-319-95269-7 (cit. on pp. 54, 60, 64, 65).

Bibliography

- [1] Tom Wijman. *The Games Market and Beyond in 2021: The Year in Numbers*. [Online; accessed 22-December-2021]. 2021. URL: <https://newzoo.com/insights/articles/the-games-market-in-2021-the-year-in-numbers-esports-cloud-gaming> (cit. on pp. 24, 57).
- [2] Cristiano Politowski et al. “Are game engines software frameworks? a three-perspective study”. In: *Journal of Systems and Software* 171 (2021), p. 110846 (cit. on pp. 24, 28, 30).
- [3] Henrik Engström et al. “Game development from a software and creative product perspective: A quantitative literature review approach”. In: *Entertainment Computing* 27 (2018), pp. 10–22 (cit. on pp. 24, 26, 27, 29).

- [4] Chris Lewis and Jim Whitehead. “The whats and the whys of games and software engineering”. In: *Proceedings of the 1st international workshop on games and software engineering*. 2011, pp. 1–4 (cit. on pp. 24, 29, 30).
- [5] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901 (cit. on pp. 24, 26, 27, 29–33, 40, 42, 43, 45, 46, 57–59, 61, 62, 73).
- [6] Jussi Kasurinen, Maria Palacin-Silva, and Erno Vanhala. “What Concerns Game Developers? A Study on Game Development Processes, Sustainability and Metrics”. In: *2017 IEEE/ACM 8th Workshop on Emerging Trends in Software Metrics (WETSoM)*. 2017, pp. 15–21 (cit. on pp. 24–26, 29, 30).
- [7] Jussi Kasurinen. “Games as software: Similarities and differences between the implementation projects”. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*. 2016, pp. 33–40 (cit. on pp. 24, 28–30, 66).
- [8] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan 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 (cit. on pp. 25, 26, 29, 30).
- [9] Jussi Kasurinen and Kari Smolander. “What do game developers test in their products?” In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 2014, pp. 1–10 (cit. on pp. 25, 26, 29).
- [10] Joe Nandhakumar, Nikiforos S Panourgias, and Harry Scarbrough. “From knowing it to “getting it”: Envisioning practices in computer games development”. In: *Information Systems Research* 24.4 (2013), pp. 933–955 (cit. on pp. 25, 26, 28–30).
- [11] Timothy McKenzie, Miguel Morales Trujillo, and Simon Hoermann. “Software engineering practices and methods in the game development industry”. In: *Extended Abstracts of the Annual Symposium on Computer-Human*

Interaction in Play Companion Extended Abstracts. 2019, pp. 181–193 (cit. on pp. 25, 26, 29).

- [12] Juergen Musil et al. “A survey on the state of the practice in video game software development”. In: *Institute of Software Technology and Interactive Systems, Tech. Rep.* (2010), p. 13 (cit. on pp. 25, 26, 29, 30).
- [13] Cristiano Politowski et al. “Are the old days gone? A survey on actual software engineering processes in video game industry”. In: *Proceedings of the 5th International Workshop on Games and Software Engineering*. 2016, pp. 22–28 (cit. on p. 26).
- [14] Cristiano Politowski et al. “Game industry problems: An extensive analysis of the gray literature”. In: *Information and Software Technology* 134 (2021), p. 106538 (cit. on p. 26).
- [15] Cristiano Politowski, Fabio Petrillo, and Yann-Gaël Guéhéneuc. “A survey of video game testing”. In: *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE. 2021, pp. 90–99 (cit. on p. 26).
- [16] Cristiano Politowski et al. “Dataset of video game development problems”. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. 2020, pp. 553–557 (cit. on p. 26).
- [17] Gabriel C Ullmann et al. “Video game project management anti-patterns”. In: *Proceedings of the 6th International ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation*. 2022, pp. 9–15 (cit. on p. 26).
- [18] Fabio Petrillo and Marcelo Pimenta. “Is agility out there? Agile practices in game development”. In: *Proceedings of the 28th ACM International Conference on Design of Communication*. 2010, pp. 9–15 (cit. on pp. 26, 29).
- [19] Michael Washburn Jr et al. “What went right and what went wrong: an analysis of 155 postmortems from game development”. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. 2016, pp. 280–289 (cit. on pp. 26, 27).

- [20] Wilson K Mizutani, Vinícius K Daros, and Fabio Kon. “Software architecture for digital game mechanics: A systematic literature review”. In: *Entertainment Computing* 38 (2021), p. 100421 (cit. on pp. 26, 27).
- [21] Jose Ricardo Mello Viana et al. “A systematic review on software engineering in pervasive games development”. In: *2014 Brazilian Symposium on Computer Games and Digital Entertainment*. IEEE. 2014, pp. 51–60 (cit. on pp. 26, 27).
- [22] Meng Zhu and Alf Inge Wang. “Model-driven game development: A literature review”. In: *ACM Computing Surveys (CSUR)* 52.6 (2019), pp. 1–32 (cit. on pp. 26, 27).
- [23] Cristiano Politowski, Yann-Gaël Guéhéneuc, and Fabio Petrillo. “Towards automated video game testing: still a long way to go”. In: *Proceedings of the 6th International ICSE Workshop on Games and Software Engineering: Engineering Fun, Inspiration, and Motivation*. 2022, pp. 37–43 (cit. on pp. 26, 27).
- [24] Ann Osborne O’Hagan, Gerry Coleman, and Rory V O’Connor. “Software development processes for games: A systematic literature review”. In: *European Conference on Software Process Improvement*. Springer. 2014, pp. 182–193 (cit. on pp. 26, 27).
- [25] Juan A Vargas et al. “A systematic mapping study on serious game quality”. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. 2014, pp. 1–10 (cit. on pp. 26, 27).
- [26] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed. “Game development software engineering process life cycle: a systematic review”. In: *Journal of Software Engineering Research and Development* 4.1 (2016), pp. 1–30 (cit. on pp. 26–30).
- [27] Saiqa Aleem, Luiz Fernando Capretz, and Faheem Ahmed. “A digital game maturity model (DGMM)”. In: *Entertainment Computing* 17 (2016), pp. 55–73 (cit. on p. 28).

- [28] Klara Borowa, Andrzej Zalewski, and Adam Saczko. “Living With Technical Debt—A Perspective From the Video Game Industry”. In: *IEEE Software* 38.06 (2021), pp. 65–70 (cit. on p. 28).
- [29] Jussi Kasurinen, Andrey Maglyas, and Kari Smolander. “Is requirements engineering useless in game development?” In: *Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings* 20. Springer. 2014, pp. 1–16 (cit. on p. 28).
- [30] Faheem Ahmed et al. “Open source computer game application: An empirical analysis of quality concerns”. In: *Entertainment Computing* 21 (2017), pp. 1–10 (cit. on p. 28).
- [31] Christopher M Kanode and Hisham M Haddad. “Software engineering challenges in game development”. In: *2009 Sixth International Conference on Information Technology: New Generations*. IEEE. 2009, pp. 260–265 (cit. on pp. 28, 29).
- [32] David Callele, Eric Neufeld, and Kevin Schneider. “Requirements engineering and the creative process in the video game industry”. In: *13th IEEE International Conference on Requirements Engineering (RE’05)*. IEEE. 2005, pp. 240–250 (cit. on pp. 29, 30).
- [33] Luca Pascarella et al. “How is video game development different from software development in open source?” In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE. 2018, pp. 392–402 (cit. on pp. 29, 30).
- [34] Barbara Kitchenham and Stuart Charters. “Guidelines for performing systematic literature reviews in software engineering”. In: (2007) (cit. on pp. 31, 34, 40, 70).
- [35] Pearl Brereton et al. “Lessons from applying the systematic literature review process within the software engineering domain”. In: *Journal of systems and software* 80.4 (2007), pp. 571–583 (cit. on p. 31).

- [36] Barbara Kitchenham et al. “Systematic literature reviews in software engineering—a systematic literature review”. In: *Information and software technology* 51.1 (2009), pp. 7–15 (cit. on pp. 31, 71).
- [37] Vander Alves et al. “Requirements engineering for software product lines: A systematic literature review”. In: *Information and Software Technology* 52.8 (2010), pp. 806–820 (cit. on p. 31).
- [38] Barbara Kitchenham. “Procedures for performing systematic reviews”. In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26 (cit. on p. 31).
- [39] Association for Computing Machinery. *The 2012 ACM Computing Classification System*. [Online; accessed 07-June-2023]. 2012. URL: <https://www.acm.org/publications/class-2012> (cit. on pp. 33, 59).
- [40] Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. “Research in software engineering: an analysis of the literature”. In: *Information and Software technology* 44.8 (2002), pp. 491–506 (cit. on p. 33).
- [41] Claes Wohlin et al. *Experimentation in Software Engineering*. Springer, 2000 (cit. on p. 33).
- [42] Michael Felderer and Guilherme Horta Travassos. *Contemporary Empirical Methods in Software Engineering*. Springer, 2020 (cit. on p. 33).
- [43] Ke Mao et al. “A survey of the use of crowdsourcing in software engineering”. In: *Journal of Systems and Software* 126 (2017), pp. 57–84. ISSN: 0164-1212 (cit. on p. 34).
- [44] Tore Dyba, Torgeir Dingsoyr, and Geir K Hanssen. “Applying systematic reviews to diverse study types: An experience report”. In: *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE. 2007, pp. 225–234 (cit. on p. 36).
- [45] Barbara Kitchenham and Pearl Brereton. “A systematic review of systematic review process research in software engineering”. In: *Information and software technology* 55.12 (2013), pp. 2049–2075 (cit. on pp. 36, 70).

- [46] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. “Guidelines for conducting systematic mapping studies in software engineering: An update”. In: *Information and software technology* 64 (2015), pp. 1–18 (cit. on pp. 36, 70).
- [47] Claes Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014, pp. 1–10 (cit. on p. 38).
- [48] Meng-Tzu Cheng et al. “The use of serious games in science education: a review of selected empirical research from 2002 to 2013”. In: *Journal of computers in education* 2.3 (2015), pp. 353–375 (cit. on p. 39).
- [49] Georgios N Yannakakis and Julian Togelius. *Artificial intelligence and games*. Vol. 2. Springer, 2018 (cit. on p. 40).
- [50] Barbara A Kitchenham et al. “An Evaluation of Quality Checklist Proposals—A participant-observer case study”. In: *13th International Conference on Evaluation and Assessment in Software Engineering (EASE) 13*. 2009, pp. 1–10 (cit. on p. 40).
- [51] Matthias Galster et al. “Variability in software systems—a systematic literature review”. In: *IEEE Transactions on Software Engineering* 40.3 (2013), pp. 282–306 (cit. on pp. 40, 41).
- [52] Muhammad Sarmad Ali et al. “A systematic review of comparative evidence of aspect-oriented programming”. In: *Information and software Technology* 52.9 (2010), pp. 871–887 (cit. on p. 40).
- [53] Tore Dybå and Torgeir Dingsøy. “Empirical studies of agile software development: A systematic review”. In: *Information and software technology* 50.9-10 (2008), pp. 833–859 (cit. on p. 40).
- [54] Anthony Fung. “The impact of the rise of mobile games on the creativity and structure of the games industry in China”. In: *Mobile Gaming in Asia: Politics, Culture and Emerging Technologies* (2017), pp. 91–103 (cit. on p. 59).

- [55] Óliver Pérez Latorre et al. “Indie or mainstream? Tensions and nuances between the alternative and the mainstream in indie games”. In: *Anàlisi* (2016), pp. 15–30 (cit. on p. 59).
- [56] Mike McShaffry. *Game coding complete*. Cengage Learning, 2009 (cit. on p. 62).
- [57] Fanny Vaudour and Aleksej Heinze. “Software as a service: Lessons from the video game industry”. In: *Global Business and Organizational Excellence* 39.2 (2020), pp. 31–40 (cit. on p. 66).
- [58] Marcos Silvano Orita Almeida and Flavio Soares Correa da Silva. “A systematic review of game design methods and tools”. In: *Entertainment Computing-ICEC 2013: 12th International Conference, ICEC 2013, São Paulo, Brazil, October 16-18, 2013. Proceedings 12*. Springer, 2013, pp. 17–29 (cit. on p. 67).
- [59] Coral Calero, Macario Polo, and M^a Ángeles Moraga. “Investigating the impact on execution time and energy consumption of developing with Spring”. In: *Sustainable Computing: Informatics and Systems* 32 (2021), p. 100603 (cit. on p. 69).
- [60] Thelma Elita Colanzi et al. “The symposium on search-based software engineering: Past, present and future”. In: *Information and Software Technology* 127 (2020), p. 106372 (cit. on p. 69).
- [61] Francisca Pérez et al. “Empowering the Human as the Fitness Function in Search-Based Model-Driven Engineering”. In: *IEEE Transactions on Software Engineering* 01 (Oct. 2021), pp. 1–1. ISSN: 1939-3520 (cit. on p. 69).
- [62] Claes Wohlin, Martin Höst, and Kennet Henningsson. “Empirical research methods in software engineering”. In: *Empirical methods and studies in software engineering*. Springer, 2003, pp. 7–23 (cit. on pp. 69, 70).
- [63] Claes Wohlin et al. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012 (cit. on p. 69).

- [64] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. *Evidence-based software engineering and systematic reviews*. Vol. 4. CRC press, 2015 (cit. on p. 70).
- [65] Stijn Shuermans and Christina Voskoglou. *Global developer population report 2019*. <https://www.slashdata.co/?link=%20GlobalDevPop19>. [Online; accessed 13-December-2022]. 2019 (cit. on p. 73).

Chapter 2

Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering

Video games pose different challenges during development and maintenance than classic software. For example, common and widespread assets, that are typically created as part of video game development are Non-Player Characters (NPCs). NPCs contribute to different aspects such as storytelling and user experience, and they are typically controlled by the CPU. We theorize that a reproduction of the actions of NPCs within the game (i.e., simulations) holds key information for Game Software Engineering (GSE) tasks such as Traceability Link Recovery (TLR). This paper presents our approach for supporting TLR in GSE by leveraging video game simulations. Simulation data from NPCs is used to reduce the search space. Since the reduced search space might still be too large for manual inspection, an evolutionary TLR procedure evolves a population of code fragments. As a result, a ranking of code fragments that map the requirement to the code is obtained. We evaluate our approach in Kromaia, a commercial video game released on PC and PlayStation 4. We compare our approach against a baseline that does not incorporate simulations by means of a statistical analysis. Our approach reduces the search space by 99.21% on average, and significantly outperforms the baseline with large differences in all performance indicators. A focus group with professional developers has confirmed the acceptance of our approach. Our work provides a new direction in TLR, which is an essential task in not only GSE but also in classic software engineering.

Author's version of the article: J. Verón, R. Lapeña, C. Cetina, Óscar Pastor, and F. Pérez, "Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering", in Lecture Notes in Computer Science, 2024, vol. 14663 LNCS, pp. 352–367, doi: 10.1007/978-3-031-61057-8_21.

2.1 Introduction

Video games are complex products where art and software are combined during the development process to conform the final product. Due to their nature and the software artifacts that compose them, video games present challenges that differ from those of classic software, leading to important differences between Game Software Engineering and Classic Software Engineering [1, 2]. Our hypothesis is that it is possible to leverage the particularities and specific domain assets of video games to provide semi-automated support and solutions for Game Software Engineering tasks.

To that extent, we have set our sight on one of the most common and widespread assets of video games: non-player characters (NPCs). NPCs are non-playable entities and elements (typically controlled by the CPU) that are created as part of the video game development process. Some examples of NPCs in video games are: the generals of the enemy troops in a Real-Time Strategy (RTS), or the rival drivers in a racing game.

It is important to note that NPCs are not created with the intention of performing Game Software Engineering (GSE) tasks on the game. Nonetheless, NPCs often have pre-programmed behaviors to allow for CPU control. We theorize that it is possible to observe NPC behaviors within the game (i.e., simulations) to reproduce and even generate knowledge that enables automated support and solutions for GSE tasks (e.g., testing the implementation of a specific requirement, or the solution to a known bug in the game). Specifically, we put the focus on Traceability Link Recovery (TLR). TLR is a software engineering task that deals with automated identification and comprehension of dependencies and relationships between software artifacts [3]. Establishing and maintaining traceability links has proven to be a time-consuming, error-prone, and labor-intensive task [3].

In this paper, we propose a novel approach that leverages the information provided by NPC simulations to provide semi-automated support for TLR between the main artifacts of GSE: the requirements and the code of a video game. To do this, the NPC's scenarios are reproduced to obtain simulation data, which is then used to reduce the search space

to a subset of code that represents the scenario (and in turn contains the requirement). Since the reduced search space might still be too large for manual inspection, an evolutionary TLR procedure along with the requirement evolves a population of code fragments. The output is a ranking of code fragments that serve as candidate solutions towards the mapping of the requirement to the code.

We evaluate the performance of our approach considering an industrial case study, which belongs to a commercial video game, Kromaia. Kromaia is a video game about flying and shooting with a spaceship in a three-dimensional space. It was released on PC, PlayStation and has been translated to eight different languages. In addition, we compare our results with those of a baseline that does not incorporate simulations within the TLR process by means of a statistical analysis and an effect size measure. Finally, we carried out a focus group interview with the aim of acquiring qualitative data and feedback about the obtained results. Hence, the contributions of this paper are threefold:

- We investigate the use of NPC simulations within the TLR process in an industrial case study. To the best of our knowledge, this is the first effort in the literature. This is relevant because TLR is an essential software maintenance and evolution task [4].
- We experimentally demonstrate that NPC simulations reduced the search space (by 99.21% on average). This enables our approach to significantly outperform the traceability results of the baseline with large differences in all performance indicators (by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure).
- We provide evidence that software engineers prefer the results of our approach using simulations over the results of the baseline even though they have not used simulations for TLR before. In fact, software engineers valued very positively both the use and the usefulness of simulations to find requirements in other video game genres and more complex problems (e.g., fighting games).

In the remainder of the paper, Section 2.2 presents the background for our work. Section 2.3 presents our approach, and Section 2.4 describes

the evaluation. Section 2.5 presents the results, and Section 2.6 discusses the outcomes of our work. Section 2.7 presents the threats to the validity. Section 2.8 reviews the related work. Finally, Section 2.9 concludes the paper.

2.2 Background

The case study we use to evaluate TLR with NPCs is the commercial video game Kromaia. Kromaia is a commercial video game programmed in C++, where the human player (i.e., the user) moves a spaceship in a three-dimensional space and fights different enemy elements through a collection of levels. In each level, the player must make the spaceship fly from an initial point to a destination point multiple times without being destroyed. The spaces where this action is carried out include: a scene or architecture with asteroids to avoid while flying, different improvements for the player's spaceship, and NPCs acting as enemies of the player (which will try to destroy the player's spaceship and have to be fought with projectiles that the player's spaceship can fire).

Video games can be defined as a series of requirements that define the multiple mechanics, dynamics, and situations in the gameplay. An example of a requirement of this nature in Kromaia is "When the human unit is damaged its armor level decreases and the interface shows the information". Developers often need to trace these requirements to fix errors or change some behaviors.

Kromaia, like most video games, has a significant amount of source code: a total of 145915 lines. Traceability between requirements and lines of code in a case like Kromaia would require approximately one hour per requirement. A game like Kromaia usually needs to trace five or more requirements every day. Thus, tracing the regular number of requirements in a regular working week would take 25 hours of work, which is impossible for a small studio to bear.

2.3 Our approach

Fig. 2.1 shows an overview of our approach. Our approach starts from a requirement that needs to be traced to the code of the game. Software engineers build a scenario for simulation that contains the requirement and use NPCs to reproduce the scenario. The simulation gathers data, registering the code that was executed to fulfill the scenario. The simulation data is used to filter the entirety of the code of the game, thus reducing the search space to a subset of code that represents the simulated scenario, which in turn contains the requirement. For the requirement “When the human unit is damaged its armor level decreases and the interface shows the information”, a scenario must be built with a single NPC that damages the human unit. This action decreases the armor level and triggers the logic to display the change in the armor level information on the graphical user interface. The logic meeting this requirement is executed and registered. Thus, this registered code constitutes the reduced search space instead of the entire source code.

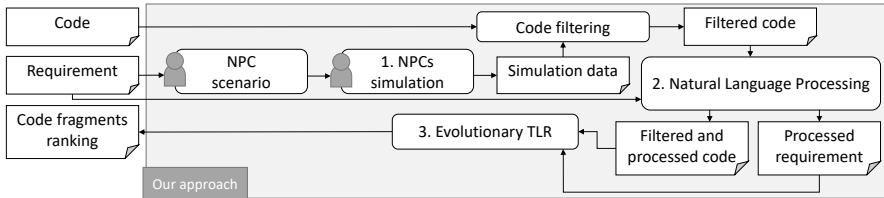


Figure 2.1: Approach overview

Since the code search space for the scenario can still be too large for manual inspection, the filtered code is then processed via Natural Language Processing (NLP) techniques. Afterwards, the filtered and processed code is fed into an evolutionary TLR procedure along with the requirement, which is also processed via NLP techniques. The evolutionary TLR procedure generates, evaluates, and evolves a population of code fragments, converging into a ranking of code fragments that constitutes the final output of the approach. Each code fragment serves as a candidate solution towards the traceability of the requirement.

The ranking of code fragments is sorted according to the parameters of the TLR fitness function, which is used within the evolutionary TLR procedure. Video game software engineers can use the ranking as a starting point to determine and document the traceability, validating or enhancing the top solutions according to their knowledge of the video game.

2.3.1 NPC-based simulations

Non-player characters (NPCs) are video game entities that simulate human players when performing specific actions in the game through various artificial intelligence algorithms created by the developers. NPCs are developed and included in a game and accompany the player during the game, antagonize the player, or simply populate the world recreated in the video game.

Nonetheless, NPCs and their simulation capabilities are embedded within the architecture of the game. Our rationale is that leveraging NPC-based simulations can support software engineering tasks in the game. Specifically, we aim to reproduce particular requirements with these simulations and capture the code executed to fulfill these requirements. This approach intends to identify code fragments associated with each requirement, thus reducing the search space.

To achieve this, the developers of the game set a scenario within the game, using its architecture and NPCs. The NPCs perform the actions defined in each requirement, substituting the human player when necessary. We register the executed code, creating filtered code documents for the requirements that must be traced. The filtered code documents are used as the search space for an evolutionary algorithm that performs the traceability procedure on the requirements.

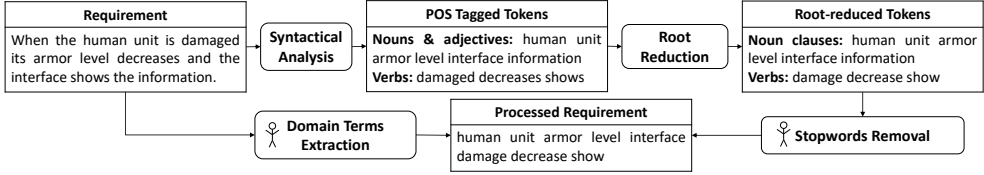


Figure 2.2: Natural Language Processing Techniques

2.3.2 Natural Language Processing

We process the input requirements and the filtered code through Natural Language Processing (NLP) techniques. This processing step serves as a means of unifying the language of the software artifacts, which facilitates the TLR process. The techniques in use are syntactical analysis, root reduction, and human NLP. Fig. 2.2 shows the used NLP techniques on a requirement in our approach.

- 1 Syntactical Analysis:** SA techniques analyze the specific roles of each word in a sentence, determining their grammatical function. These techniques filter words that fulfill specific grammatical roles in a requirement, discarding those that do not have semantic value (such as articles or adverbs).
- 2 Root Reduction:** The technique known as lemmatizing reduces words to their semantic roots (lemmas), thus avoiding verb tenses, noun plurals, and several other word forms that can negatively interfere with the TLR process.
- 3 Human NLP:** Human NLP is often carried out through domain term extraction or stopwords removal. Our approach searches the requirements for domain terms provided by software engineers and adds the domain terms found to the processed artifact. On the other hand, stopwords are filtered out after root reduction, using a list of stopwords that is also provided by the software engineers.

2.3.3 Evolutionary TLR

Fig. 2.3 shows an overview of the Evolutionary Algorithm that generates, evaluates, and evolves a population of code fragments, converging into a ranking of potential solutions towards the input requirement. The algorithm takes both the filtered and processed code, and the processed requirement as input, and iterates over a population of code fragments, modifying them using genetic operations until a final ranking is produced as output. More precisely, the algorithm runs in three steps:

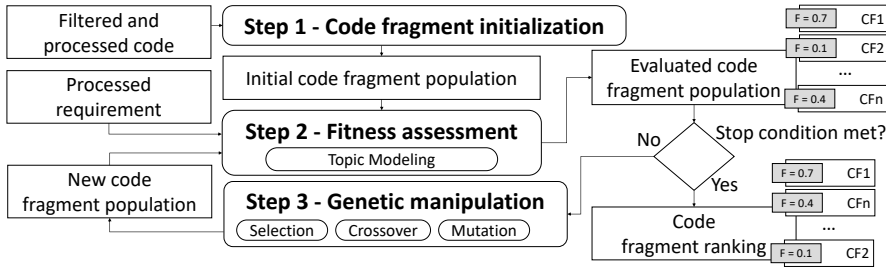


Figure 2.3: Evolutionary algorithm

1. **Initialization:** The algorithm generates a population of code fragments that serves as input for the algorithm. To generate the population, parts of the code documents are extracted randomly and then added to a collection of code fragments, which are linked to the code documents they belong to.
2. **Fitness function:** This step assesses each of the candidate code fragments produced, ranking them according to a fitness function. Our approach uses Topic Modeling as fitness function because it obtained the best results when performing Feature Location, a similar Information Retrieval task to the one at hand [5].

To do this, we utilize Latent Dirichlet Allocation (LDA) [6] since it is one of the most popular topic modeling methods [7]. LDA is an unsupervised probabilistic technique for estimating a topic distribution over a text corpus made up of a set of documents, where each document is a set of terms. As a result, a probability distribution is obtained for each document, indicating the likelihood that it expresses each topic. In addition, a probability distribution is

obtained for each topic identified by LDA, indicating the likelihood of a term from the corpus being assigned to the topic.

LDA inputs include: the documents (D), the number of topics (K), and a set of hyper-parameters. The hyper-parameters of any LDA implementation are: k , which is the number of topics that should be extracted from the data; α , which influences the topic distributions per document. A lower α value results in fewer topics per document; and β , which affects the distribution of terms per topic. A lower β value results in fewer terms per topic, which in turn implies an increase in the number of topics needed to describe a particular document.

LDA outputs include: ϕ , which is the matrix that contains the term-topic probability distribution; and θ , which is the matrix that contains the topic-document probability distribution.

To assess the relevance of each code fragment with regard to the provided requirement (i.e., query), we use LDA. Given the terms for the query Q and the outputs of LDA (ϕ and θ), the conditional probability P of Q given a document D_i is computed as follows [8]:

$$Sim(Q, D_i) = P(Q|D_i) = \prod_{q_k \in Q} P(q_k|D_i) \text{ where } q_k \text{ is the } k^{th} \text{ pro-}$$

cessed term in the query, and D_i is a document (i.e., a code fragment) of the population that is made up of a set of processed terms.

Fig. 2.4 shows an example of the fitness assessment for each candidate code fragment (i.e., individual) from the population. Given the processed terms of the requirement (query) and each code fragment (represented as a column in the matrix of the figure), the figure depicts how a value is assigned per term in each code fragment of the population (values in the cells of the matrix) by using the LDA outputs (the ϕ and θ matrices). For example, the conditional probability for the processed term *human* given the code fragment CF_n , that is, $P(human|CF_n) = 0.39$ (as can be seen in the shaded cell in the matrix of Fig. 2.4). Using the values of the matrix, the fitness value is obtained for each code fragment as shown in the vector of

Fig. 2.4. The highest fitness value in the vector of the figure is 0.41, which belongs to CF_2 .

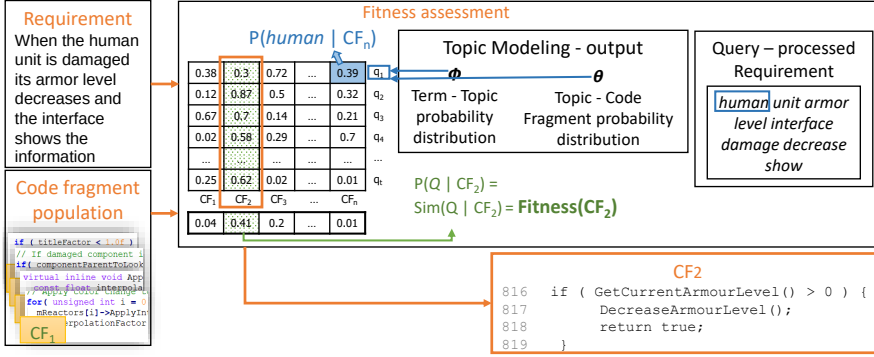


Figure 2.4: Example of the fitness assessment of code fragments

3. **Genetic manipulation:** Finally, if the solution does not converge, this step generates a new population of code fragments through genetic manipulation. The generation of new code fragments, based on existing ones, is done by applying a set of three genetic operations (selection, crossover, and mutation), which are taken from the literature [9]:

The **selection operation** picks candidates from the population as input for the rest of the operations. Candidates with high fitness values have higher probabilities of being chosen as parents for the next generation. The **crossover operation** enables the creation of a new individual by combining genetic material from two parent code fragments. The **mutation operation** imitates mutations that randomly occur in nature when new individuals are born, by adding or removing code lines from the fragment.

Step 2 (fitness assessment) and Step 3 (genetic manipulation) are repeated until the solution converges to a certain stop condition (e.g., number of iterations). When this occurs, the evolutionary algorithm provides a code fragment ranking, ordered according to the values determined by the fitness function.

Similarly to other works that retrieve text from an initial query using LDA or other information retrieval techniques, the results depend on the quality of the queries [8], which is typically improved through an iterative refinement process [10]. Therefore, even when irrelevant code fragments are obtained in the ranking, the results can be considered as a starting point for the iterative refinement process. From there, software engineers can either manually modify the proposed ones or modify the requirement to automatically obtain different code fragments for the solution.

2.4 Evaluation

2.4.1 Research Questions

From our work, several research questions arise:

- RQ₁** *Does the inclusion of simulations influence the traceability search space in Game Software Engineering?*
- RQ₂** *What is the performance in terms of solution quality of the traceability results in Game Software Engineering for our approach using simulations and a state-of-the-art technique?*
- RQ₃** *Are the differences in performance between our approach using simulations and the baseline significant?*
- RQ₄** *How much do the simulations influence the quality of the solution compared to the baseline?*

2.4.2 Experimental Setup

The inputs to build the test cases are the requirements and the code. These test cases are then employed to perform TLR in both our approach and the baseline. The baseline is similar to our approach and it also uses topic modeling as a fitness function (as described in Section 2.3.3), but, unlike our approach, it does not integrate simulations into the TLR process and it does not reduce the search space. Finally, a measurements report is obtained after comparing the approved trace-

ability that is provided as input (used as oracle) with the results of the baseline and our approach.

The case study that we used in the evaluation is Kromaia, a commercial video game which code has been provided by our industrial partner, Kraken Empire¹. Kromaia is made up of 145915 lines of code. In total, 18 requirements with their corresponding oracles and simulation scenarios were provided by the company. The provided requirements were selected and provided by the developers, who considered those requirements as being a representative collection in terms of maintenance. The approved traceability for each requirement (i.e., oracle) is the set of code lines (i.e., code fragment) that correspond to the full coverage for a requirement in the traceability search space. The code fragment that corresponds to each requirement has between 1 and 43 code lines.

As a result of comparing the approved traceability with the results of the baseline and our approach, a measurements report is obtained to answer each research question as follows.

Answering RQ₁: To determine whether the inclusion of simulations influences the traceability search space in GSE, the evaluation starts by taking the inputs of the case study (requirements, code, and simulation scenarios and approved traceability) to obtain the set of methods that are called during the simulations scenarios for each test case. This set of methods will conform the traceability search space for each test case. Afterwards, it is possible to assess whether the size of the traceability search space has been reduced, and if so, by how much.

In addition, the traceability search space is compared to the approved traceability to determine whether the traceability search space limits the quality of the solution by excluding code lines that are in the approved traceability.

Answering RQ₂: To determine the performance in terms of solution quality for each test case in our approach and the baseline, we compare the code fragment that achieves the highest fitness score in our approach

¹<https://www.krakenempire.com/>

and in the baseline against its respective oracle (i.e., ground truth) to calculate a confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model, comparing a set of test data (the solutions provided by the approach) against a set of known true values (the solutions from the oracle). In our case, the solutions obtained by the approach are code fragments, which are composed of a subset of code lines that are part of the original code. Since the granularity is at the level of code lines, the presence or absence of each code line is considered as a classification. The confusion matrix distinguishes between the predicted values (i.e., solution of the approach) and the real values (i.e., solution of the oracle), classifying them into four categories: (1) True Positive (TP), values that are predicted as true and are true in the real scenario; (2) False Positive (FP), values that are predicted as true but are false in the real scenario; (3) True Negative (TN), values that are predicted as false and are false in the real scenario; and (4) False Negative (FN), values that are predicted as false but are true in the real scenario.

From the values in the confusion matrix, it is possible to derive a series of performance measurements. Specifically, we report three performance measurements that are widely accepted by the software engineering community [11]: recall, precision, and F-measure. Recall ($\frac{TP}{TP+FN}$) measures the number of elements of the solution that are correctly retrieved by the proposed solution (how many code lines from the oracle are present in the retrieved code fragment). Precision ($\frac{TP}{TP+FP}$) measures the number of elements from the solution that are correct according to the ground truth, i.e., the oracle (how many code lines from the retrieved fragment appear in the oracle). Finally, F-measure ($2 * \frac{Precision * Recall}{Precision + Recall}$) corresponds to the harmonic mean of precision and recall.

For each requirement in both the baseline and the approach, we executed 30 independent runs (as suggested by Arcuri and Fraser [12]): 18 (requirements) x 2 (baseline and approach) x 30 repetitions, for a total of 1080 independent runs.

Answering RQ₃: To determine whether the differences in performance between our approach and the baseline are significant, the results must

be properly compared and analyzed using statistical methods. With the aim of providing formal evidence that the differences do in fact have an impact on the performance measurements, we follow the guidelines presented in [13]. The statistical test that must be followed depends on the properties of the data, and it is accepted by the research community that a *p-value* under 0.05 implies statistical significance [13]. For each performance measure a *p-value* is recorded.

Answering RQ₄: To determine how much the performance is influenced by using our approach compared to the baseline, it is important to assess whether our approach is statistically better than the baseline, and if so, to measure by how much. To do this, we use Cliff’s delta [14], which is an ordinal statistic that describes the frequency with which an observation from one group is higher than an observation from another group compared to the reverse situation. It can be interpreted as the degree to which two distributions overlap, with values ranging from -1 to 1. For instance, when comparing distributions of the treatment and the control, a value of 0 means no difference between the two distributions, a value of -1 means that all of the samples in the distribution of the treatment are lower than all of the samples in the distribution of the control, and a value of 1 means the opposite. In addition, threshold values can be defined [15] for the interpretation of Cliff’s delta effect size as *negligible* ($|d| < 0.147$), *small* ($|d| < 0.33$), *medium* ($|d| < 0.474$), and *large* ($|d| \geq 0.474$). A Cliff’s delta value is recorded for each pair-wise comparison between our approach and the baseline for each performance measure.

2.4.3 Implementation details

For the development of our approach and the baseline, we used Eclipse with Java. For the NLP operations used in both our approach and the baseline, we used the OpenNLP Toolkit [16].

Topic Modeling was implemented using the Collapsed Gibbs Sampling (CGS) for LDA because it requires less computational time [17], and it was previously used for locating features in source code [8].

Since the focus of this paper is not to tune the values to improve the performance of our approach when applied to a specific problem, we used default values from the literature [9, 5, 18]. As suggested by Arcuri and Fraser [12], default values are sufficient to measure the performance of search-based software engineering algorithms.

For the evaluation, we used an Asus ROG Strix G15 G512LW-HN038 laptop with an Intel(R) Core(TM) i7-10750H processor, 16GB RAM, an NVIDIA® GeForce® RTX 2070 graphics card, and Windows 11 (64-bit). The oracle was provided directly by the developers of the video game.

The CSV files used as input in the statistical analysis as well as an open-source implementation of the approach are available here: <https://github.com/VeronLinks/simulations-ss-tlr>

2.5 Results

2.5.1 *Research Question 1*

After obtaining the set of methods that are called during simulation scenarios for each test case, the traceability search space has been reduced from 145915 code lines to an average of 1157 code lines. Afterwards, the traceability search space is checked to determine whether any of the code lines that are included in the oracle have been omitted. If no code line has been omitted, the quality of the solution of our approach is not limited after reducing the search space.

RQ₁ answer: The inclusion of simulations has been shown to significantly reduce the search space for TLR in GSE by 99.21% on average. We have found that the inclusion of simulations does not limit the quality of the solution.

2.5.2 Research Question 2

Table 2.1 shows the mean values and standard deviations of recall, precision, and F-measure for our approach and the baseline. As the values show, our approach outperforms the baseline in all performance measurements.

Table 2.1: Mean values and standard deviations for recall, precision, and the F-measure

	Recall $\pm (\sigma)$	Precision $\pm (\sigma)$	F-measure $\pm (\sigma)$
Our work	65.58 $\pm$ 24.32	76.36 $\pm$ 14.90	67.65 $\pm$ 15.43
Baseline	42.35 $\pm$ 25.71	58.02 $\pm$ 8.35	44.78 $\pm$ 16.84

RQ₂ answer: The results reveal that our approach outperforms the baseline in the three performance measurements. Hence, the inclusion of simulations improves the quality of the traceability results in GSE when using a state-of-the-art technique by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure.

2.5.3 Research Question 3

The p – values obtained for our approach are 0.0037 for recall, 9.4^{-6} for precision, and 4.1^{-5} for F-measure.

RQ₃ answer: Since the Quade test p -values are smaller than the 0.05 statistical significance threshold for all performance measurements, we can state that there are significant differences in performance between our approach and the baseline for TLR in GSE.

2.5.4 Research Question 4

With regard to how much the simulations influence the quality of the solution compared to the baseline, the obtained Cliff’s Delta values of the three reported performance measurements are large, being 0.5 for recall, 0.7623 for precision, and 0.6512 for F-measure.

RQ₄ answer: From the effect size analysis of the Cliff’s Delta values that are obtained when our approach is compared to the baseline, we can conclude that there is a large influence on the quality of the solution for

all performance measurements with our approach using simulations for TLR in GSE.

2.6 Discussion

Industrial video games feature many requirements involving multiple methods, but they also bring opportunities to TLR. Simulations, scarce in traditional software, are common in video games due to NPCs being created during the development. By analyzing the results, the noise that the simulations remove by reducing the search space helps the evolutionary TLR procedure to work significantly better than when using the complete code.

At least in the case study that was used in this work, there is no limitation in the quality of the solution (recall) when the reduced traceability search space is obtained. Nevertheless, our approach should be replicated with case studies from other domains before assuring the generalization of the results. Nevertheless, our results suggest that leveraging video game simulations for TLR is promising.

Our focus group, consisting of five software engineers in the field of GSE, included one with 15 years of video game development experience, two with six years, and two with two years of experience. This focus group dealt with the following open questions: (1) What do you think of the results of the approaches?; (2) How do you feel about locating requirements in video games using simulations?; (3) How do you imagine locating requirements using simulations in video games of other genres and in more complex video games?

The software engineers valued very positively the results of our approach using simulations, and stated that they preferred our approach using simulations because the results were far superior to the results of the baseline. Although none of the five engineers had thought of or used simulations for TLR before, they agreed on the relevance of simulations for TLR in video games. In their opinion, the idea of using simulations for TLR is promising, and they would use simulations for TLR in video games on a daily basis.

The engineers imagined using our approach in another of their video games since they could also use simulations of NPCs to find requirements that give them trouble. In addition, the engineers confirmed that our approach using simulations can be used for other video game genres and more complex problems.

2.7 Threats to validity

In this section, we acknowledge the limitations of our approach using the classification of threats to validity proposed by Wohlin et. al. [19]:

Conclusion validity: To minimize this threat, we have based our work on research questions. In addition, the requirements and code used in our approach were taken from an existing commercial video game. None of the authors of this work were involved in the creation of the game or in the generation of the data, which was provided by the developers of the game.

Internal Validity: To minimize this threat, we have used the same NLP techniques on all of the software artifacts and followed the same evaluation process for all evaluated approaches. In addition, the available test cases represent a broad scope of different scenarios within the case study in an accurate manner.

Construct validity: To minimize this threat, our evaluation was performed using three measurements that are widely accepted and utilized in the state-of-the-art literature: precision, recall, and F-measure. Moreover, we have used the same software artifacts for all of the approaches, representing the same scenarios, so that cause and effect are equally represented among them.

External Validity: This threat is concerned with generalization. All of the artifacts under study in our work (requirements and code) are frequently used in video games development. In addition, TLR is a common practice in all kinds of video game development scenarios and the real-world case study used in our research represents the industrial scene of video game developments well. Moreover, we defined our approach inde-

pendently of the case study, and then applied it to the case study. Hence, our approach can potentially work in any video game scenario where requirements and code are available. Nevertheless, our results should be replicated with other case studies before ensuring their external validity.

2.8 Related work

After reviewing a traceability survey that contains works from 1999 to 2011 [20], we used the same query presented in the survey, and we found 11 more works between 2012 and November 2023. For example, Dekhtyar et al. [21] assess the accuracy of provided Requirements Traceability Matrix (RTM) by means of an assessment committee composed of three to five different trace recovery methods, such as Probabilistic Information Retrieval (ProbIR). Parvathy et al. [22] propose automating the computation of traceability by looking at the correlation between documents. Gethers et al. [23] propose the the Relational Topic Model for TLR and combine different orthogonal IR-based methods for improving recovery accuracy. These three approaches require additional information sources beyond source code, such as RTM, design documents, and domain models that support traceability. However, in the development of commercial video games, creating or synchronizing these artifacts is often neglected. Therefore, approaches like this work that focus on searching for solutions in terms of code fragments in the source code are necessary in this context. Lapeña et al. [24, 25] leverage an already existing feature of the software artifacts to improve TLR using BPMN models and leveraging on execution traces. None of these works use simulations to reduce the search space.

We also performed a manual search and we only found three studies on locating artifacts in video games for GSE. Two of the three studies focus on bug location using simulations, but neither reduces the search space as we propose. Blasco et al. [9] focus on TLR for video games and use an evolutionary algorithm for obtaining a code fragment from the source code that realizes a requirement specified in a natural language. Casamayor et al. [26] use simulations in video games and leverage the existence of NPCs in video games to remove any additional cost for the

developers, as we do, but they do not use these simulations to reduce the search space. Prasetya et al. [27] present an agent-based automated testing framework for locating bugs in video games.

The rationale behind using simulations is that video game bugs are not always related to code malfunction, making traditional testing and bug location inefficient. Video games bugs include other aspects related to design, such as balancing certain values. Testing and bug location in video games need to be reactive and able to interpret their environment to determine what actions to take. This rationale differs from the rationale behind using simulations in this work. We leverage these simulations for performing automated and precise executions of the program, and registering the executed code. Thus, the search space for TLR can be reduced with the goal of improving the quality of the results.

2.9 Conclusion

In this work, we have focused on GSE and TLR. GSE is a very novel field of work that deals with the challenge of developing and maintaining the software of video games, whereas TLR is a key support activity for the development, management, and maintenance of software. Video games not only bring challenges to TLR, but they also provide opportunities such as the information that can be extracted from the simulations of NPCs, which are created as part of the video game.

This information had not yet been exploited for either reducing the search space or for TLR. We have filled this gap, and we have shown that the search space can be reduced and that TLR can be significantly improved by leveraging video game simulations in a commercial video game. Specifically, compared to the baseline, the search space was reduced by 99.21% on average, and our approach improved the results with large differences (by 23.23% in recall, 18.34% in precision, and 22.87% in F-measure). A focus group has demonstrated that software engineers prefer the results of our approach using simulations over the results of the baseline. The software engineers have also highlighted the usefulness of simulations to find requirements in other video game genres and more complex problems (e.g., fighting games).

Our work opens a new direction in TLR, which is an essential task in both GSE and in classic software engineering. Other essential maintenance tasks such as bug location (our future work) can also benefit from the use of simulations.

Acknowledgements

Partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by Gobierno de Aragón (Research Group S05_20D), and by Centro para el Desarrollo Tecnológico y la Innovación (CDTI) (Reference: SAV-20221011).

Bibliography

- [1] Jorge Chueca et al. “The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”. In: *Information and Software Technology* (2023) (cit. on p. 98).
- [2] Apostolos Ampatzoglou and Ioannis Stamelos. “Software engineering research for computer games: A systematic review”. In: *Information and Software Technology* 52.9 (2010), pp. 888–901 (cit. on p. 98).
- [3] Rocco Oliveto et al. “On the equivalence of information retrieval methods for automated traceability link recovery”. In: *IEEE 18th International Conference on Program Comprehension*. 2010, pp. 68–71 (cit. on p. 98).
- [4] Manuel Ballarín et al. “On the influence of architectural languages on requirements traceability”. In: *Software: Practice and Experience* 53.3 (2023), pp. 704–728. DOI: <https://doi.org/10.1002/spe.3166>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3166> (cit. on p. 99).
- [5] Francisca Pérez et al. “Topic modeling for feature location in software models: Studying both code generation and interpreted models”. In: *Information and Software Technology* 140 (2021), p. 106676. ISSN: 0950-5849 (cit. on pp. 104, 111).

- [6] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. “Latent dirichlet allocation”. In: *J. Mach. Learn. Res.* 3 (2003). ISSN: 1532-4435. DOI: <http://dx.doi.org/10.1162/jmlr.2003.3.4-5.993> (cit. on p. 104).
- [7] Hamed Jelodar et al. “Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey”. In: *Multimedia Tools and Applications* 78 (2019), pp. 15169–15211. DOI: 10.1007/s11042-018-6894-4 (cit. on p. 104).
- [8] Lauren R. Biggers et al. “Configuring Latent Dirichlet Allocation Based Feature Location”. In: *Empirical Softw. Engg.* 19.3 (June 2014), pp. 465–500. ISSN: 1382-3256 (cit. on pp. 105, 107, 110).
- [9] Daniel Blasco, Carlos Cetina, and Óscar Pastor. “A fine-grained requirement traceability evolutionary algorithm: Kromaia, a commercial video game case study”. In: *Information and Software Technology* 119 (2020), p. 106235 (cit. on pp. 106, 111, 115).
- [10] Emily Hill, Lori Pollock, and K. Vijay-Shanker. “Automatically Capturing Source Code Context of NL-queries for Software Maintenance and Reuse”. In: *Proceedings of the 31st International Conference on Software Engineering*. ICSE ’09. 2009, pp. 232–242. ISBN: 978-1-4244-3453-4 (cit. on p. 107).
- [11] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986. ISBN: 0070544840 (cit. on p. 109).
- [12] Andrea Arcuri and Gordon Fraser. “Parameter tuning or default values? An empirical investigation in search-based software engineering”. In: *Empirical Software Engineering* 18.3 (2013), pp. 594–623. ISSN: 1573-7616. DOI: 10.1007/s10664-013-9249-9 (cit. on pp. 109, 111).
- [13] Andrea Arcuri and Lionel Briand. “A Hitchhiker’s Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering”. In: *Softw. Test. Verif. Reliab.* 24.3 (May 2014), pp. 219–250. ISSN: 0960-0833. DOI: 10.1002/stvr.1486 (cit. on p. 110).
- [14] Norman Cliff. *Ordinal methods for behavioral data analysis*. Lawrence Erlbaum Associates, Inc, 1996 (cit. on p. 110).

- [15] Jeanine Romano et al. “Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen’sd for evaluating group differences on the NSSE and other surveys”. In: *annual meeting of the Florida Association of Institutional Research*. 2006, pp. 1–33 (cit. on p. 110).
- [16] The Apache Software Foundation. *Apache OpenNLP: Toolkit for the processing of natural language text*. <https://opennlp.apache.org/>. 2020 (cit. on p. 110).
- [17] Cam-Tu Nguyen Xuan-Hieu Phan. *JGibbLDA. A Java Implementation of Latent Dirichlet Allocation (LDA) using Gibbs Sampling for Parameter Estimation and Inference*. 2020 (cit. on p. 110).
- [18] Jaime Font et al. “Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering”. In: *IEEE Transactions on Evolutionary Computation* PP.99 (2017), pp. 1–1. ISSN: 1089-778X (cit. on p. 111).
- [19] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on p. 114).
- [20] Markus Borg, Per Runeson, and Anders Ardö. “Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability”. In: *Empirical Software Engineering* 19 (2014), pp. 1565–1616 (cit. on p. 115).
- [21] Alex Dekhtyar et al. “Technique integration for requirements assessment”. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE. 2007, pp. 141–150 (cit. on p. 115).
- [22] Anju G Parvathy, Bintu G Vasudevan, and Rajesh Balakrishnan. “A comparative study of document correlation techniques for traceability analysis”. In: *International Conference on Enterprise Information Systems*. Vol. 3. SCITEPRESS. 2008, pp. 64–69 (cit. on p. 115).
- [23] Malcom Gethers et al. “On integrating orthogonal information retrieval methods to improve traceability recovery”. In: *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*. 2011, pp. 133–142 (cit. on p. 115).

- [24] Raúl Lapeña et al. “Leveraging execution traces to enhance traceability links recovery in BPMN models”. In: *Information and Software Technology* 146 (2022), p. 106873 (cit. on p. 115).
- [25] Raúl Lapeña et al. “Leveraging BPMN particularities to improve traceability links recovery among requirements and BPMN models”. In: *Requirements Engineering* (2022), pp. 1–26 (cit. on p. 115).
- [26] Rodrigo Casamayor et al. “Bug localization in game software engineering: evolving simulations to locate bugs in software models of video games”. In: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. 2022, pp. 356–366 (cit. on p. 115).
- [27] ISWB Prasetya et al. “An agent-based approach to automated game testing: an experience report”. In: *Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation*. 2022, pp. 1–8 (cit. on p. 116).

Chapter 3

A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD

Video game engines are used in most modern video games because they simplify and speed up development. In addition, some of the most popular engines, such as Unreal Engine 5 (UE5), also integrate visual scripting tools. Visual scripting in UE5, through Blueprints, is a model-driven development approach that replaces text code, like C++, with a visual language of interconnected nodes representing functions and data flows, forming a flowchart-like logic diagram. This approach simplifies game development by abstracting complex code into intuitive, visual models, enabling creators to construct and iterate game components without extensive programming knowledge. Although Blueprint models usually decrease the complexity of implementing components, thus accelerating the development, they might lead to less energy-efficient runtime performance than C++. In this work, we evaluate the energy consumption of three relevant video game components (health system management, inputs processing, and collections operations for an inventory), each implemented with Blueprint models and C++. The results show that the energy consumption per frame when using C++ is up to 48% lower than when using Blueprint models. The combination of artistic and technical profiles in video game developments has favoured the adoption of Blueprint models. However, there is a lack of works analyzing the energy consumption. Until this work, there was no evidence that the success of models for developing video games, like the one under study in this work, was accompanied by a cost in energy consumption for certain situations. Given the huge popularity of video games, this cost in energy might reach up to the equivalent of the energy consumption of 28 million European households.

Author's version of the article: J. Verón, C. Pérez, C. Calero, M.Á. Moraga, F. Pérez, and C. Cetina, "A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD", in MODELS '24: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 114 - 125, doi: 10.1145/3640310.3674099.

3.1 Introduction

Video game engines integrate many different assets for the development of video games, like a graphics engine or a physics engine, as well as many tools that wrap around them to accelerate development. The video game engine enables developers to create different games in a much more agile way, without having to repeat or code low-level elements whose design from scratch would be very costly and incompatible with the pace of development of these products. Although there are companies that develop their video game engines, it is more common to use an already existing engine from the market. Unreal Engine [1] is one of the most popular choices among developers.

One essential component of video game engines is software models. Developers have the option to create video game content either through direct coding (e.g., C++) or by utilizing the software models provided by the engines. While coding offers developers greater control over the content, software models provide a higher level of abstraction, distancing themselves from the underlying implementation and technology. This enables developers to work with concepts more closely related to the problem domain. Consequently, developers are freed from the intricate details of common low-level implementations like those of physics and graphics, allowing them to focus on the content of the game. In 2014, Unreal Engine included its own domain-specific modeling language called Blueprints. Ten years after the launch, the adoption of Blueprints by developers is majority [2, 3, 4].

The gaming industry is steadily gaining prominence in the realm of computer-based activities. The evolution of recent video games is marked by, among other things, enhanced graphical realism and an increase in the number of interactions and players, needing more powerful equipment capable of managing this complexity without compromising user experience. This gain in computational complexity usually implies that the energy consumption and the environmental footprint of video games are also increasing.

An approximate estimation of the global video game energy consumption can be made by multiplying the number of worldwide video game players

(3.38 billion according to [5]) times the average consumption per player. This later amount is obtained by multiplying the average hours a gamer plays per year (440 hours [6]) times an estimation of the power required by a representative video game, which we obtain from our measurements of the average power (358.6 W) required when playing *Baldur's Gate III* without PC background processes. To produce a more conservative estimation, we do not include console or mobile gamers in this estimation, only PC gamers are considered, which account for 43% of the total players [7]. As a conclusion, we have estimated the energy consumption of video games per year to be at least 230 TWh worldwide. This estimation assumes that all gamers played the same game. This figure is equivalent to the annual energy consumption of approximately 59 million European households (3900 kWh is the average annual amount of electricity purchased by an EU member residential electric-utility customer [8]). That means that improvements in video game software have enormous potential for energy savings. However, perhaps due to its relative youth, the video game sector has not yet developed the same level of environmental awareness as other computing technology sectors.

This work constitutes the first approach to evaluate the difference in energy consumption between using Blueprint models and using a more traditional coding approach in C++ for a video game. For enhancing result reliability, we employed the validated framework FEETINGS [9] for the measurement and analysis of the energy consumption, following the validated procedure included in the framework (GSMP) and employing the technological infrastructure that facilitates the capture and analysis of software energy consumption, which are accepted by the green computing research community [10, 11, 12].

We studied three components of video games in two different versions (using Blueprint models for one version and C++ code for the other one) to measure and evaluate their energy consumption. These three components are health system management, input processing, and collections operations for an inventory. Industry professionals have confirmed them as relevant in video games. Both versions of each component have also been developed by one company and then revised by other industry professionals.

The results show that C++ can have up to 48% less energy consumption than Blueprint models for the execution of some components. Software models (such as Blueprints) are proving to be a success in the gaming domain, but this success is accompanied by an unidentified energy consumption problem that this work reveals.

This work has the potential to encourage video game engine developers to improve the generation of code from models from a greener perspective. Furthermore, these results might also encourage a new branch of research on green model-driven video game development. Finally, this work raises awareness on the adoption of models being accompanied by green challenges, and this phenomenon is not only limited to the success of models in video games, but can also take place in more traditional software domains.

The paper is structured as follows: in Section 3.2 we provide a concise summary of video game engines, focusing on Unreal Engine 5 and how Blueprint models are integrated to foster video game development. In Section 3.3 we discuss how the energy consumption measurement approach is planned and performed following accepted practices by the green research community. In Section 3.4 we introduce the Research Questions and explain the implementation details. In Section 3.5 we analyze and discuss the results. The threats to the validity of the experiment are discussed in Section 3.6. In Section 3.7 we outline previous related work on the subject. Lastly, we present our summarized conclusions and future work in Section 3.8.

3.2 Background

A game engine is a tool in the field of video game development, encapsulating a suite of integrated technologies and tools that streamline the production of games. It not only expedites the development process, but also enhances the quality and performance of the final product, thereby playing a crucial role in the evolution and proliferation of video games as a popular form of entertainment and artistic expression.

Game engines are specifically designed to enable and streamline the creation of video games, providing a layer of software components that are reusable across different video games, thus significantly reducing the time and resources required to develop a new game from scratch. These engines can be equipped with a wide array of functionalities including, but not limited to, rendering graphics (2D and 3D), simulating physics, managing memory and assets, and facilitating audio output, networking capabilities for multiplayer games, artificial intelligence for non-player character behavior, and tools for animation and environmental effects.

Additionally, many game engines offer more advanced features such as multi-platform building, which makes them an appealing choice if it is the case of a video game or project targeted for multiple platforms. With this feature, game engines allow game developers to focus on the design and gameplay aspects of the game rather than the intricacies of platform-specific development of these components.

Most game developers use game engines, (60% as of 2022 [13]) and there are multiple industry-scale engines. One of the most popular game engines among video game developers is Unreal Engine [13]. Video game developers have used Unreal Engine since its launch in 1998 until the current version of Unreal Engine 5 (UE5). Nowadays, UE5 is the preferred choice for high budget developments. In fact, nearly 80 of the most highly anticipated games released in 2023 are powered by Unreal Engine [14].

It is important to take into account that UE5 is not only one of the most used game engines among video games developers, but also millions of users play to video games powered by UE5: the video game *Fortnite* has more than 235 million users per month [15], and it is only one of the many games developed with this engine. Furthermore, the adoption of UE5 includes a wide range of applications beyond traditional gaming, including film and television, architecture, automotive, or manufacturing, and simulation [16], although their main focus is video game development.

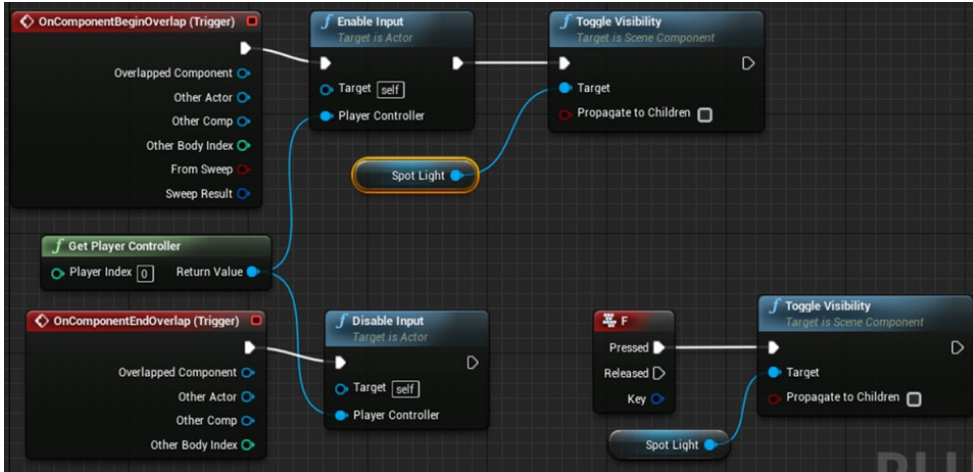


Figure 3.1: Blueprint models in UE5: Interactive Spotlight Control via Trigger

3.2.1 Unreal Engine 5 Blueprints

The software model of UE5 (a domain-specific modeling language) utilizes a node-based language called Blueprints, as depicted in Figure 3.1, where functions and actions are represented as interconnected nodes. It's crucial to distinguish between the concept of software model and that of a 3D model (or mesh). 3D models are commonly employed in the fields of video games and computer graphics to denote the visual representation of three-dimensional shapes.

Blueprints stands out as a feature that democratizes the game development process, allowing programmers and designers to create complex game logic and interactions without the need for traditional code-based programming.

In the development of video games, a seamless fusion of artistry and programming expertise is essential, resulting in a product that reflects the collaborative endeavor of development teams conformed by diverse profiles. According to a recent industry survey [17], these teams are formed by software developers (24%), game designers (23%), artists (15%), UI designers (8%), and QA engineers (5%). The diversity within these teams fosters a predilection for utilizing software models. Software models offer

a higher level of abstraction than coding, thus facilitating greater involvement from members less focused on technical aspects. This strategy not only democratizes the development process but also enriches the creative input across various roles.

The Blueprint in Figure 3.1 is designed to control input and visibility of a spot light, based on the player's overlapping with a trigger area. When the character controlled by the player enters the trigger area, input is enabled, allowing them to interact with the spot light by pressing the specified key 'F'. When the key is pressed, the visibility of the spot light is toggled, meaning it will turn on or off. Exiting the trigger area disables the input, preventing the player from toggling the visibility of the spot light until they re-enter the trigger area.

This logic implementation approach significantly lowers the entry barrier for game development, enabling individuals with limited coding knowledge to contribute effectively to the development process instead of needing to learn how to program in C++, which is the alternative and more traditional option for logic implementation in UE5. Moreover, Blueprints facilitate rapid prototyping and iteration, as changes can be made and tested in real time, thus accelerating the development cycle and encouraging experimental gameplay designs.

Blueprints need to be compiled into UnrealScript VM bytecode which will run on a virtual machine when used in-game [18]. The difference in the logic of the game between C++ code and Blueprint models might not be noticeable in terms of behavioural results; however, the impact difference in terms of energy consumption between using C++ or Blueprint models has not been explored so far.

3.3 Studying the energy consumption of software

If we want to rigorously study the environmental impact of software, it is essential to know its energy consumption while in operation as precisely as possible.

Two kinds of approaches serve this purpose: The first one uses specialized software tools capable of estimating hardware energy consumption during software execution, such as PowerAPI [19] or Joulemeter [20]. These tools do not directly measure energy consumption but instead derive estimates from a model fed with parameters collected by the software during execution. Despite their ease of use and ability to yield valuable results across various levels of detail, their reliance on estimation implies that result accuracy eventually depends on the model approximations.

Alternatively, the other approaches involve employing measuring equipment linked to a suitably instrumented computer to directly record energy consumption during software execution. This approach offers the advantage of delivering genuine energy consumption data, whose quality only depends on equipment quality and measurement protocol. Nonetheless, it requires computer instrumentation for testing and may prove to be more expensive and difficult to implement.

Merely possessing a physical device to measure computer energy consumption during software operation is not adequate to ensure conclusive results. A comprehensive process is imperative to ensure study rigor and consistency, thereby rendering obtained conclusions to be scientifically relevant.

In this paper, we have employed the Framework for Energy Efficiency Testing to Improve Environmental Goals of the Software (FEETINGS) [9]. This framework is accepted by the green computing community [10, 11, 12] and offers the following assets:

- A standardized terminology for measuring software energy consumption.
- A green software measurement process (GSMP), accessible as an electronic guide (<https://alarcos.esi.uclm.es/FEETINGS/>), designed to systematically support the necessary tasks for measuring software energy consumption.
- A technological infrastructure that facilitates the capture and analysis of software energy consumption. This infrastructure includes

an Energy Efficiency Tester (EET), serving as a measuring instrument, and a software tool (ELLIOT) designed for visualizing and analyze the energy consumption results captured by the EET automatically. The EET, a hardware-based device, has been utilized instead of software estimators to provide more realistic and accurate energy consumption values. It captures consumption data directly from the power supply of the PC, offering a sampling frequency of 100 Hz. The EET measures the consumption of the entire PC, referred to as the Device Under Test (DUT), as well as various components of the PC, including the processor, graphics card, hard disk, and monitor.

Monitor	Falkon Q2702S 27" 2K
Motherboard	Asus Prime B460-Plus
Processor	i7 10700
RAM	4 modules of 32GB DDR4 Kingston 2666MHz CL16
Graphics card	Zotac Gaming GForce RTX 3060 12 GB GDDR6
Hard Disk	Hard Disk & Kingston SSD A400 – 480GB SATA
Power supply	Energy PS901SX 900W
O.S.	Windows 11 Pro

Table 3.1: DUT specifications

The GSMP process comprises seven phases, covering all the essential steps for conducting a comprehensive analysis of software energy consumption during software execution:

1. **Scope definition.** The main goal of this phase is to obtain a complete specification of the requirements for the evaluation of energy efficiency, including the precise definition of the software to be analyzed and the development of the different test cases to be run. The test cases for our work are explained in detail in Section 3.4.1.

Our study aims to compare the energy consumption of three components representative of real video games, implemented with two different approaches (Blueprint models and C++ code). It is worth emphasizing that our objective is to find out if there is any significant difference in energy consumption, not to discover what this

difference is exactly, as this will always depend on the environment in which the applications are run.

2. **Measurement environment setting.** This phase includes activities such as the selection of the measurement equipment to be used, with the detail of its technical specifications; the decision of the physical magnitudes to be measured, and the verification that there are no background processes that may contaminate the desired results.

The measurement environment used in this work is the offered one in the FEETINGS framework, which includes: the EET device, the ELLIOT software, and the DUT. The DUT is a desktop computer utilized for executing the test cases and conducting time and power measurements. In our experiment, the DUT had the specifications outlined in Table 3.1.

In green software research, accepted metrics include the energy use of the processor, the hard disk drive (HDD), and the total energy consumption of the device [10]. However, assessing the energy use of the graphics card becomes essential for video games, given its intensive utilization in gaming.

To take care of the possible contamination of background processes necessarily running in the operative system, the final power measurements are acquired by subtracting the baseline power required (measured before launching each application under study) from the measurements taken during the experiment (measured while executing each application).

3. **Measurement environment preparation.** This stage includes the elimination of services or processes running in the background of the DUT, the setting of the sample size (since measurements must be repeated to guarantee the representativeness of the results), and the configuration of the testbed.

The EET replaces the original power source of the DUT where the software runs with its own power supply. This way, the power required and the execution time are measured and recorded. To guar-

antee the reliability of the analysis and the statistical results obtained, all the test cases executions were recorded 30 times. Also, in our study, the duration of the execution and recording was 30 seconds in all of the test cases at a sampling frequency of 100 Hz and the execution of every test case did not require any actions performed by a human, as the actions of each test case are performed automatically.

4. **Performing the measurements.** Measurements are performed in the sequence defined in the previous phase, and all the raw data are collected and recorded.
5. **Test case data analysis.** It covers the analysis of the previous data, checking for incorrect values and eliminating them from the dataset, and performing the statistical analysis of the data.

A measurement is considered invalid if it is recorded from a wrong execution, identified by its inconsistency with the results from the other executions; or if it qualifies as an outlier, where one or more values are significantly higher or lower than those from other executions. This processing is computed by ELLIOT, which also calculates the statistics of the values obtained.

6. **Software entity data analysis.** It includes the calculation of the energy consumed by the execution of the software by subtracting the appropriate baseline consumption, and the analysis and interpretation of the results, to make sure they address the research questions that were put forward at the beginning of the research.

In this work, the analysis and interpretation of the data are presented in Section 3.4.2 and Section 3.5.

7. **Reporting the results.** Finally, phase 7 deals with appropriate reporting of the exercise, to ensure that the gained knowledge is available and both the measurement process and results can be replicated by other researchers.

3.4 Evaluation

This section includes the research questions that we aim to answer, the implementation details, and the results obtained.

This study aims to examine the impact on the energy consumption of two different approaches for implementing logic in a video game (i.e. Blueprint models and C++), but video games integrate many complex components to create immersive and interactive experiences. If we evaluated the consumption of a complete video game, we would not be able to isolate specific components, as video game scenes often combine many of these at once. These components encompass graphics rendering, physics engines, audio processing, health system management, input processing, and inventory¹ management, among many others.

Some of these components, like graphics rendering, physics, and audio processing, are already managed by game engines themselves, such as UE5. However, some other components need to be implemented by the game developers. This is the case of health system management, input processing, and inventory operations. These components are also three of the most common components in video games, which makes them representative cases for the implementation comparison in this work.

An example of the usage of an inventory system is shown in Figure 3.2. The inventory screen in this Figure displays items which the players can use on their character. On the left side, materials for different interactions like crafting and making and modifying equipment are displayed. The center shows the weapons of the character, like swords and a crossbow, along with some number consumable objects and other fighting elements like bombs. The right side showcases the current armor of the character and other equipment, as well as the health status. Between the center and the right side, there is a representation of the character with the currently equipped weapons and armor. The inventory helps players to manage resources and gear within the game.

¹Inventories are collections of game elements that the player can manage. This results in the need of implementing different operations, including search operations and memory management.



Figure 3.2: Inventory system in the video game *The Witcher® 3: Wild Hunt*

An example of the usage of a health system and input processing is shown in Figure 3.3. This Figure shows the main character, controlled by the player, in a natural setting. The red bar at the top left is the health bar, indicating the life energy of the character. When this bar empties, the character can no longer continue and the game is restarted from a previously saved point. The player moves and orders the character through this virtual world using various inputs from a keyboard or a controller, as indicated by the buttons on the bottom right of the screen. These would correspond to buttons on a game controller for actions like moving, attacking, or dodging. Both Figure 3.2 and Figure 3.3 are taken from the industrial case *The Witcher® 3: Wild Hunt*, a blockbuster video game which had sold more than 50 million units by the first quarter of 2023 [21].

We aim to compare the energy consumption of video games with a different logic implementation approach: one using Blueprint models and the other one using C++ programming. In order to tackle this, we formulated the following Research Questions:

- RQ1 Does implementing a health system management component for a video game with Blueprint models entail more energy consumption at run time than with C++?



Figure 3.3: Health bar at the top left of the image and main character moving with the input of the player in the centre of the image in the video game *The Witcher® 3: Wild Hunt*

RQ2 Does implementing an input processing component for a video game with Blueprint models entail more energy consumption at run time than with C++?

RQ3 Does implementing an inventory search component for a video game with Blueprint models entail more energy consumption at run time than with C++?

Answering each of the Research Questions will lead us to compare the energy consumption difference between the two different logic implementations of each of the three components, allowing us to conclude if there is a difference in energy consumption between using Blueprint models and using C++ programming or not.

3.4.1 Implementation Details

For this study, we need six test cases that allow us to compare the measurements taken from both logic implementation approaches under study, i.e., we had to implement three components in two different versions each: one version using Blueprint models and the other version using C++ code. The components implemented are the following: health system

management, input processing, and inventory operations. The outcome of implementing each component in both versions is a total of six test cases (three of them implemented with Blueprint models and the other three in C++).

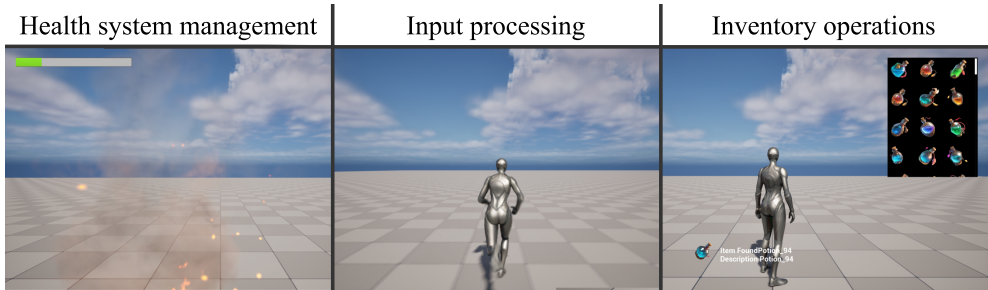


Figure 3.4: The three components implemented in UE5: health system management, input processing, and inventory operations.

Currently, there are no publicly available video games with two versions which are identical in every way except for the implementation approach used (Blueprint models or C++ code). Therefore, the implementations were performed in collaboration with a professional video game development company. The components were implemented by the company using the latest version (5.3.2) of Unreal Engine, which in turn used the version 17.6.5 of Microsoft Visual Studio Community 2022. Each version of each component was implemented by the professional developers in the company and then reviewed by other professional developers unaffiliated to the company, who agreed on the way to develop it.

The six applications were compiled in release mode with default shipping settings for Windows, since it is the way that would reach end users and a video game is expected to be run a much higher number of times by all end users than by developers. Also, the six applications built in UE5 are not running only the component under study. The applications require a minimum scenario for the execution of the components to be as similar as possible to how these components would appear in an industrial case. The minimum scenarios have other components working; like a physics system, or graphics rendering; in order to conform a complete application built with UE5. All of these components, except for the one under study,

are similar between the applications containing each of the two versions of the components under study.

Figure 3.4 shows a screenshot of each one of the three different scenarios, each containing one of the components under study. Some of the additional components can be seen in this Figure, such as a character (grey humanoid) and an environment (sky and floor).

For the implementation of the health system management component, displayed in the left image of Figure 3.4, the character enters a damaging zone and its health gets decreased in time. In the Blueprint models version, the variables for the maximum and current health points are assigned and set in a Blueprint class named *AC_HealManager*. Then, another Blueprint class named *DamageZone* reduces the current health in *AC_HealManager* over time when the character is situated in the set damaging area. In the C++ version, the same logic design is followed in the C++ classes: *UHealManager* sets the maximum and current health and *ADamageZone* reduces the current health stored in *UHealManager* over time when the character is situated in the set damaging area. Snippets of both implementations approaches are shown in Figure 3.5 for this component.

For the implementation of the input processing component, displayed in the center image of Figure 3.4, the character moves when an input associated with the movement is pressed in a keyboard or a controller. The reading of the values of the inputs is implemented in both versions with the input system *Enhanced Input* of UE5, as usually done in commercial video games. In the Blueprints version, the association between the inputs values and the movement of the character is controlled in a Blueprints class named *BP_CustomPlayer*. In the C++ version, the movement of the player is controlled in the C++ class *AcustomPlayer*, which binds the input actions with the functions that make the player move. Snippets of both implementations approaches are shown in Figure 3.6 for this component.

For the implementation of the inventory operations component, displayed in the right image of Figure 3.4, both versions first load an inventory with 157 items. Then, an item is searched in that inventory periodically. In

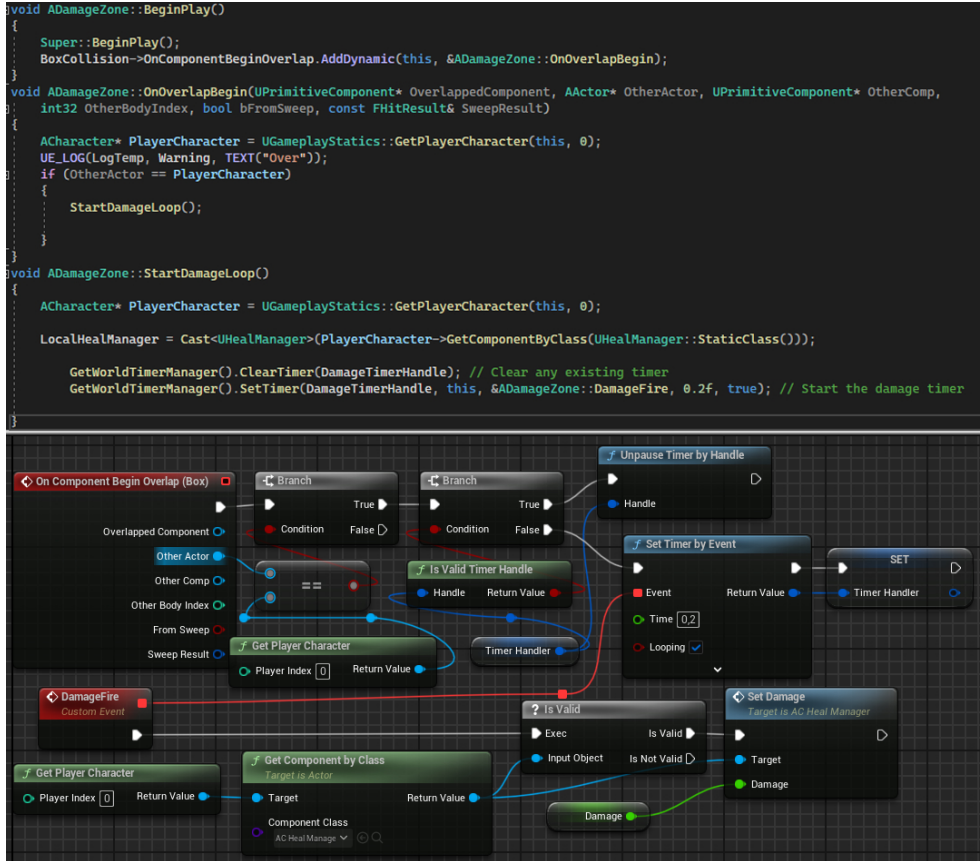


Figure 3.5: C++ and Blueprint models snippet where the character receives a decrease in current health when damaged.

the Blueprints version, every item in the inventory is an instance of a Blueprints class named *BP_ItemData* and the search in the inventory is performed by using the UE5 Blueprint node *Find*, which searches for a specific item in a Blueprint Map with a provided key and returns the item if it is found. In the C++ version, the items are instances of a struct named *FDataItem* and the search in the inventory is performed in the C++ class *UInventoryManager* by calling the function *Find* of the Unreal Engine 5 class *TMap*, which returns a pointer to the value of the element if the map contains the key. Snippets of both implementations approaches are shown in Figure 3.7 for this component.

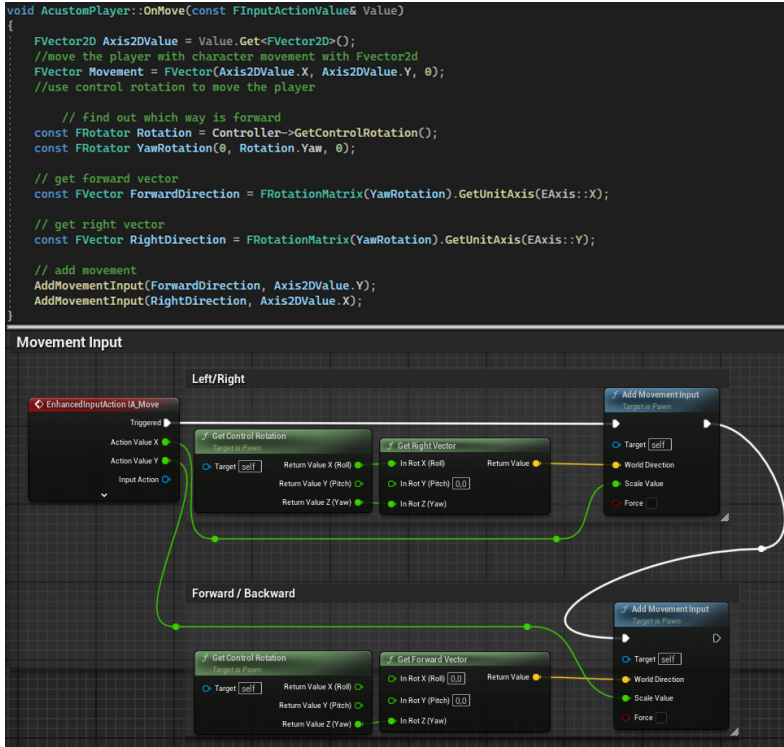


Figure 3.6: C++ and Blueprint models snippet where the input is processed for moving the character.

All Blueprints have been compiled with the default compiler

All six applications; i.e., the applications containing the two versions of each of the three components, implemented with Blueprint models and C++ code; are available at the following URL for replication purposes, along with the source code of the Unreal Engine 5.3.2 projects: <https://zenodo.org/records/12570903>

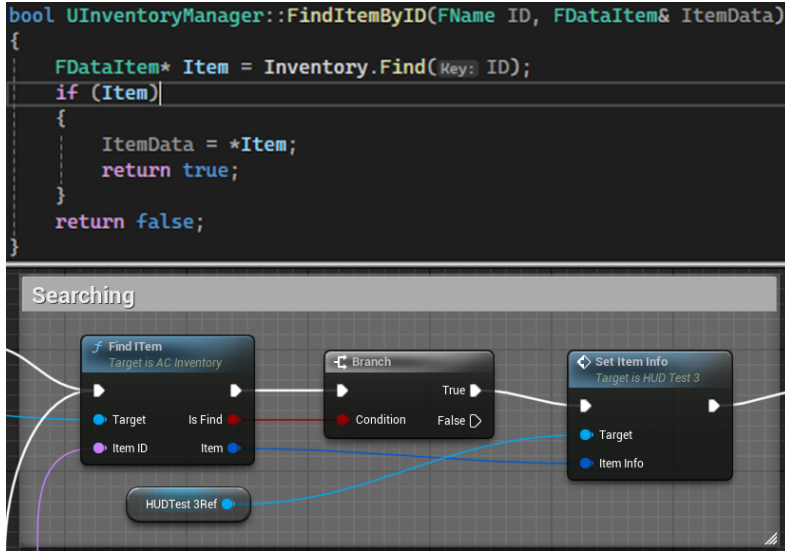


Figure 3.7: C++ and Blueprint models snippet where an item is searched in the inventory.

3.4.2 Results

Table 3.2 shows the power required for executing the six applications for 30 seconds each. These results show that differences are minimal between the usage of Blueprint models and C++ code for all three components. The power required by the hard disk is between one and two orders of magnitude lower than that of the graphics card and processor, and the difference between both implementation approaches in the graphics card and the processor is under 1%. These results entail that, if we measure the power used over time (i.e. W.s), the answers to RQ1, RQ2, and RQ3 are similar: Implementing in Unreal Engine 5 any of the three tasks under test using Blueprint models results in similar energy consumption at runtime than implementing them with C++.

When measuring traditional software, only its energy consumption on W.s is measured [10]. However, we argue that to have a better understanding of energy consumption in video games, frame rate needs to be taken in consideration. As video games are real-time applications which require continuous visual feedback for the player, refresh rate in this type of applications is key. This refresh rate is usually reflected in the

TestCase	HDD	GraphicsCard	Processor	DUT
Health system (Blueprints)	1.73	100.68	31.14	324.38
Health system (C++)	1.73	100.85	30.86	330.62
Power diff	0.03%	0.17%	-0.93%	1.89%
Input processing (Blueprints)	1.66	101.59	45.88	370.87
Input processing (C++)	1.75	102.32	45.75	376.07
Power diff	4.81%	0.71%	-0.27%	1.38%
Inventory ops. (Blueprints)	1.77	101.30	45.33	375.04
Inventory ops. (C++)	1.73	100.70	45.44	374.53
Power diff	-2.19%	-0.60%	0.24%	-0.14%

Table 3.2: Power required (W) for the execution of the applications for 30 seconds and power difference comparison (C++ - Blueprints)

frame rate or frames per second (FPS) in this context. FPS represent the amount of frames (i.e. images) that the application is able to render per second, with the consequent logic for updating the game world that is being rendered. The more FPS, the more fluid the gameplay will be, improving the user experience and the quality of the gameplay. Frame rate is so important that platforms owners like Sony or Nintendo will only allow developers to publish their games on their platforms (e.g., PlayStation 5 or Switch) if they achieve a minimum frame rate at any given time of the gameplay.

Test Case	C++ (ms)	Blueprints (ms)	Time diff.
Health system	0.78	0.82	-5.81%
Input processing	1.07	1.05	1.73%
Inventory ops.	0.66	0.98	-47.73%

Table 3.3: Average time required (ms) for the rendering of a frame of the applications and time difference comparison (C++ - Blueprints)

To take into account the FPS, we additionally measured the time in milliseconds that frames required in average during the executions of the applications described in Section 3.4.1. The results of this additional measurement, which are displayed in Table 3.3, show that the C++ implementation of the Health system component takes a 5.81% less time to render a frame, which means that the Blueprint models implementation of this component can achieve an average of 5.81% less FPS. The C++ implementation of the Input processing component, in con-

trast, takes a 1.73% more time to render a frame, which means that the Blueprints models implementation of this component can achieve an average of 1.73% more FPS. Finally, the inventory operations component is the component where the biggest difference between both implementation approaches is found: The C++ implementation of the inventory operations component takes 47.73% less time to render a frame, which means that the Blueprints models implementation of this component can achieve an average of 47.73% less FPS.

We showed in Table 3.2 the measurement of the power consumption of each implementation approach of each component in 30 seconds of execution, and in Table 3.3 the average time required for rendering a frame. With these measurements, we can compute the power consumption values per average frame. These values are displayed in mW for each version of each component are displayed in Table 3.4.

TestCase	HDD	GraphicsCard	Processor	DUT
Health system (Blueprints)	0.04	2.53	0.78	8.16
Health system (C++)	0.04	2.39	0.73	7.84
Power diff	-6.03%	-5.88%	-7.05%	-4.06%
Input processing (Blueprints)	0.05	3.28	1.48	11.96
Input processing (C++)	0.06	3.35	1.50	12.31
Power diff	6.22%	2.19%	1.21%	2.84%
Inventory ops. (Blueprints)	0.05	3.02	1.35	11.19
Inventory ops. (C++)	0.04	2.03	0.92	7.56
Power diff	-50.96%	-48.62%	-47.38%	-47.94%

Table 3.4: Power required (mW) per frame on average for the applications and power difference comparison (C++ - Blueprints)

Therefore, the actual answers to the Research Questions are the following:

Answering RQ1 Yes, implementing a health system management component for a video game with Blueprint models entails more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 5.88% less graphics card consumption and 7.05% less processor consumption than with Blueprint models.

Answering RQ2 No, implementing an input processing component for a video game with Blueprint models does not entail more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 2.19% more graphics card consumption and 1.21% more processor consumption than with Blueprint models.

Answering RQ3 Yes, implementing an inventory search component for a video game with Blueprint models entails noticeably more energy consumption at run time than with C++. Specifically, implementing this component with C++ requires 48.62% less graphics card consumption and 47.38% less processor consumption than with Blueprint models.

3.5 Discussion

To understand why there are differences in energy consumption, we analyzed the open information about the UE5 implementation. UE5 has a lot of documentation about the blueprint model classes and about the C++ classes. All the C++ and Blueprints classes used to implement the components in this paper have their documentation online. However, the analysis of what is publicly available does not provide any explanation to justify the energy differences.

Our intuition is that the differences in power consumption are caused during the compilation of the Blueprint model to code. This compilation is outlined in the Unreal Engine 5 documentation [18]. In other words, the documentation describes the main steps of the compilation but does not provide the actual transformation between Blueprint models and code. We expect that, as works like this one raise awareness about the energy consumption of video games, video game engine developers start paying attention to which parts of the model transformation are influencing the energy consumption. Possibly, opening the model transformation will also help the game engine developers community and the research community to advance on improving the energy consumption.

It is also necessary that the documentation informs developers about the energy consumption implications carried by using different Blueprint concepts (nodes such as Find Element in the Array class) for the implementation. For that purpose, it is essential to keep researching to understand the differences in the consumption of Blueprint classes and the interactions among them. Currently, there is nothing similar in the official documentation because of the current lack of awareness about the energy consumption of video games.

Currently, Unity is the other most used video game engine currently in the video game industry [13]. Unity stands out because it offers a versatile performance range tailored to various platforms, making the development of mobile and XR games more appealing compared to Unreal Engine. Unity relies on C# as programming language for video game developers, but Unity seems to be following Unreal Engine steps in the adoption of Model-Driven Development (MDD). In 2017, the company Ludiq released Bolt: a domain-specific modeling language for Unity via the Unity Store. In 2020, Unity Technologies (the developers of Unity) acquired Bolt and, a year later, in 2021, Unity released its official Unity Visual Scripting language based on Bolt. Nowadays, Unity Visual Scripting is an integral part of Unity.

Our work reveals that the success of Blueprint models for developing video games with Unreal Engine comes with a penalty in energy consumption. This suggests that Unity Visual Scripting might also require attention. We have consulted the Unity Visual Scripting documentation and found no public information on the transformation of Unity Visual Scripting to code, similar to the lack of public information on the transformation of Blueprint models.

In fact, the transformations of video game software models to code should represent an opportunity to achieve the green perspective. Model-to-code transformations can embed green best practices to ensure their utilization by construction rather than solely depending on the developer's skill. Advances in green practices could be embedded into new versions of transformations, facilitating adoption by the developer community. To achieve this, it is necessary to raise awareness of the energy consumption of video games.

With the present work, we intend not only to raise awareness of the lack of research on energy consumption in MDD for video games but also to encourage and ease other researchers to perform experiments on other engines, like Unity. We release our implementations not only for replication purposes but also for extensibility. We expect other researchers to leverage the design of the components in this work and the rationale on the importance of FPS to better understand energy consumption in video games.

3.6 Threats to validity

We use the classification of threats to validity of Wohlin et al. [22].

Construct validity: This aspect of validity reflects "the degree to which the independent and the dependent variables are accurately measured by the measurement instruments used in the experiment".

Dependent variables: The power required for executing the different test cases is taken as the dependent variable. It was measured using the Efficient Energy Tester, validated as a dependable device for assessing the energy efficiency of software execution [10]. Additionally, each execution was repeated 30 times, and all applications containing the components under study were executed on the same physical machine, operating system, and configuration.

Independent variables: The independent variables are the use of Blueprint models or C++ to implement the different behaviors within Unreal Engine 5 and the three behaviors selected as representative of a typical game which consumption is to be assessed.

Internal Validity: As we try to determine if there is a causal relation between implementation approach (Blueprint models or C++) and energy consumption, it is important to check if the factor being investigated might be influenced by unaccounted variables.

Due to the nature of the experiments, all of the variables were controlled, thereby minimizing potential threats to internal validity. Specifically, the test cases are the same and run the same components in both approaches.

It is worth noting that the repetition of the experiments 30 times during the same time also helped to minimize any possible bias. In addition, subtracting the baseline power in the measurements and removing invalid measurements (wrong executions or outliers) also helped to ensure that the dataset was valid for analyzing and answering the Research Questions.

External Validity: This aspect of validity focuses on the extent to which findings can be generalized and are relevant to other cases. Enhanced external validity indicates a greater ability to apply the results of an empirical study to real-world software engineering practices.

To minimize the risk of the Blueprints and C++ implementations not being comparable, the components have been implemented in collaboration by a professional video game development company specialized in development with Unreal Engine who are equally experienced with both C++ and Blueprints. Furthermore, the representativeness of the components as relevant components of video games that are present in the vast majority of video games being marketed today has been validated by this same company and two more additional professionals in the industry.

Another threat to the external validity is the fact that the measurements are obtained for a specific computer (the DUT being used). We mitigated this threat by employing a pre-validated procedure and measuring tool, enhancing result reliability. Even if consumption data were to vary across devices, our results indicate that the percentage variances between those values would be similar to those obtained in this case.

In addition to the classification of Wohlin et al. [22], it is important to remember that many components run simultaneously for a video game to function correctly. Although the three components under study were selected by a professional video game development company as frequently used, they may not be the most relevant in terms of power usage. Further studies with industrial case studies will be required to shed more light on this in the future.

3.7 Related Work

Projections suggest that by 2025, around 20% of global energy usage will stem from information and communication technologies [23]. Consequently, the environmental footprint of this sector is significant and poised for further expansion in the foreseeable future. One of the sectors within these technologies where growth has been most evident in recent years is video games.

However, perhaps due to its relative youth, the video game sector has not yet developed the same level of environmental awareness as other computing technology sectors. In the case of computing technologies in general, there are research efforts in the area of green cloud computing [24], green mobile computing [25], green software [26], and green data centers [27] among others.

The research discussed in the work of Gutierrez et al. [11] emphasizes two approaches to examining energy usage: Green-IN and Green-BY. The former concentrates on enhancing energy efficiency within a specific domain (such as video games), while the latter focuses on utilizing a domain (like video games) to promote sustainability across various contexts.

Within the subcategory of "green-BY", it is where the research efforts are concentrated. For example, Johnson et al. [28] conducted a systematic review to assess the effectiveness of gamification and serious games in impacting domestic energy consumption.

Within the subcategory of "green-IN" computing, the focus has mainly been on studies analyzing the impact of hardware improvements on the environment. In recent years, the impact of proper software use (green-IN software) on the environment has also been recognized, and more effort is being devoted to its study [11].

However, there are hardly any studies that analyze the energy consumption of video game software despite its importance. A survey on "green video games", which is defined as the field of Green Computing that deals with the particularities of video games, is presented in the work

of Pérez et al. [29]. Three main issues are identified in the survey by Pérez et al. [29] related to Green Video Games:

- *Battery life:* The surge of mobile devices as gaming platforms, coupled with the heightened complexity and graphic quality of applications, has underscored the constraints imposed by battery life on the gaming experience. Consequently, several studies are dedicated to enhancing the energy efficiency of video games, driven less by environmental considerations and more by the recognition of how reduced energy consumption prolongs battery life and, consequently, enhances the user experience.
- *Balance between energy consumption and various factors like application performance, user experience, and economic costs:* Several methods aimed at lowering energy usage in computer applications, notably in video games, such as Dynamic Voltage and Frequency Scaling (DVFS), may inadvertently compromise performance or user satisfaction, while others might affect economic considerations. Consequently, numerous studies concentrate on assessing the optimal parameter combination to achieve maximum energy savings with minimal deterioration in application performance, user experience, and acceptable costs.
- *Modelling and measurement of energy consumption:* Enhancing software energy efficiency requires a thorough understanding of the impact of each element on the overall energy consumption. Research endeavors aimed at quantifying software energy consumption under controlled circumstances and modeling its behavior in the pursuit of more energy-efficient versions of applications like video games.

The survey shows that there are studies on the energy consumption of video game software focusing on algorithmic efficiency. However, the survey does not identify any work that studies the influence of software models (such as Blueprints) on energy consumption as this work does.

The energy consumption of two of the most popular video game engines (Unity and Unreal Engine) is studied in the work of Pérez et al. [30], comparing their performance in three different scenarios representative of

typical video games. The authors conclude that, since each of the engines outperforms the other in energy efficiency depending on the scenario, there is interest in finding the most energy-efficient video game engine technology.

Although the comparison focuses on the influence of game engines, it does not consider the influence of video game engine software models (Blueprints in Unreal Engine) on energy consumption as this work does. Considering the influence of video game software models on energy consumption is important because (1) they are increasingly used in video game development, and (2) they have higher energy consumption than their coding alternative, as this work shows.

In the broad sense of sustainability, there is a first wave of work that discusses the role that software models can play to achieve sustainability. The Sustainability Evaluation ExperienceR (SEER) framework [31] argues that software models are one of the main enablers to achieve sustainability in the context of self-adaptive systems. Gramelsberger et al. [32] propose a DSL model to describe sustainability indicators (eg., consumption types, CO2 emissions, or landscape usage). This DSL can be combined with ADLs to facilitate informed sustainability decision-making throughout the lifecycle of systems during their development. In the context of digital twins, David and Bork [33] provide an initial taxonomy that focuses on the technical dimension of sustainability. All of these works make the case that the modeling community is relevant to tackle the challenge of sustainability. Our work goes beyond in the sense of providing the first quantitative evidence of the importance of green MDD. It is true that software models can help to achieve sustainability, but it is also true that the very software models can pose a challenge to sustainability if energy consumption goes unnoticed.

An informal survey of the papers published in the MODELS Conference Proceedings and in the SoSyM Journal with the terms *consumption OR energy OR green* shows that the topic of energy consumption has not received sufficient attention from the community. As with video game models, models of other classical domains may also influence energy consumption. This is especially relevant in the era where low-code and no-code approaches are proliferating in industry as is the case of Microsoft

Power Apps, Amazon Honeycode (recently ended or hiatus), Outsystems, or Google AppSheet.

3.8 Conclusion

The video games industry has experienced a significant expansion in user numbers and game sophistication in recent years. However, the environmental awareness in the video games industry, particularly the efforts to reduce the energy consumption, is still an under-explored field of research that requires more work on it, as the magnitude of the energy consumed in this sector makes this a relevant issue for the research community.

Unreal Engine 5 is one of the most used game engines in the industry and powers some of the most played video games in the present, which highlights its significance in this field. This engine offers two different logic implementation approaches to developers: Blueprint models and C++.

This work explores how different these implementation approaches are in terms of energy consumption. We selected three representative components to measure the differences in energy use between both implementation approaches: health system management, inputs processing, and inventory operations. The power required for the different implementation approaches has been measured in a controlled environment for all of the components. We employed the framework FEETINGS, which is accepted in the green computing community. This framework provided us the systematic process which we followed for the measurement and the technological infrastructure for capturing and analyzing the energy consumption.

When analyzing the power required per frame, we find that in two out of the three components (health system management and inventory operations) Blueprint models consume more energy per frame than C++ code. The inventory operations component is particularly interesting, showing nearly 50% less energy consumption per frame with the C++ implementation compared to Blueprint models. In the one component where C++ code consumes more power per frame (inputs processing),

the difference is between 1% and 3% for both the graphics card and the processor.

Furthermore, there are still relevant common video game components to analyze in future works, such as networking, save and load system, dialogue system, particle system, camera behaviour, and AI path-finding navigation. Also, video game engines enable deployment of video games across multiple target platforms (e.g., PC, mobile, VR, consoles), as it widens the audience reach for developers. This cross-platform development may reveal new insights into how models influence energy consumption in different devices, thus making the analysis of the same game components in different platforms another goal for our future work.

This work reveals that the success of software models in the video game domain comes with an energy cost. Due to the popularity of video games, this energy cost can be very high, equivalent to the consumption of millions of households. Another leading engine in the video game industry, Unity, is following the MDD (Model-Driven Development) steps of Unreal Engine, reinforcing the importance of considering the green perspective in MDD for video games. Moreover, the use of MDD outside the video game domain could impact energy costs in other areas. This work could prompt the video game industry and researchers to consider the green perspective and trigger a new branch on research in green MDD beyond video games.

Acknowledgements

Partially supported by MINECO under the Project VARIATIVA (PID2021-128695OB-I00), by UE under the Project OASSIS (PID2021-122554OBC31/AEI/10.13039/501100011033/FEDER), by CECD (JCCM) and FEDER funds under the Project EMMA (SBPLY/21/180501/000115), by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/ under the project SEEAT (PDC2022-133249-C31), and by MCIN/AEI/10.13039/501100011033 and European Union NextGenerationEU/PRTR under the project PLAGEMIS (TED2021-129245B-C22).

Bibliography

- [1] Epic Games Inc. *The most powerful real-time 3D creation tool - Unreal Engine*. <https://www.unrealengine.com/>. [Online; accessed 26-March-2024]. 2024 (cit. on p. 124).
- [2] Jimmy Thang. *Solo dev Gwen Frey explains how she developed puzzle game Kine using only Blueprints*. <https://www.unrealengine.com/en-US/developer-interviews/solo-dev-gwen-frey-explains-how-she-developed-puzzle-game-kine-using-only-blueprints>. [Online; accessed 26-March-2024]. 2019 (cit. on p. 124).
- [3] Nikolai Baskin Sokolov. *Unreal Engine 5: C++ vs Blueprints*. <https://gamedev.gg/c-vs-blueprint-in-unreal-5/>. [Online; accessed 26-March-2024]. 2021 (cit. on p. 124).
- [4] Brian Crecente. *Unreal Engine powers stylish stellar mystery The Invincible*. <https://www.unrealengine.com/en-US/developer-interviews/unreal-engine-powers-stylish-stellar-mystery-the-invincible>. [Online; accessed 26-March-2024]. 2023 (cit. on p. 124).
- [5] Newzoo International B.V. *Newzoo's Global Games Market Report 2023*. <https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version#:~:text=Highlights%20of%20the%20free%202023%20Global%20Games%20Market%20Report%3A&text=The%20number%20of%20players%20worldwide,year%20growth%20of%20%2B0.6%25..> [Online; accessed 16-January-2024]. 2023 (cit. on p. 125).
- [6] Jessica Clement. *Average weekly hours spent playing video games in selected countries worldwide as of January 2021*. <https://www.statista.com/statistics/273829/average-game-hours-per-day-of-video-gamers-in-selected-countries/>. [Online; accessed 19-December-2023]. 2023 (cit. on p. 125).
- [7] Newzoo International B.V. *How consumers engage with video games today*. [https://newzoo.com/resources/trend-reports/global-gamer-study-free-report-2023?utm_campaign=2023-06-GGS-GGS%202023%](https://newzoo.com/resources/trend-reports/global-gamer-study-free-report-2023?utm_campaign=2023-06-GGS-GGS%202023%20)

- 201launch%20report&utm_source=Press. [Online; accessed 16-January-2024]. 2023 (cit. on p. 125).
- [8] *Electricity consumption per dwelling*. <https://www.odyssee-mure.eu/publications/efficiency-by-sector/households/electricity-consumption-dwelling.html>. [Online; accessed 16-January-2024]. 2023 (cit. on p. 125).
- [9] Javier Mancebo et al. “FEETINGS: Framework for energy efficiency testing to improve environmental goal of the software”. In: *Sustainable Computing: Informatics and Systems* 30 (2021), p. 100558 (cit. on pp. 125, 130).
- [10] Coral Calero, Macario Polo, and M^a Ángeles Moraga. “Investigating the impact on execution time and energy consumption of developing with Spring”. In: *Sustainable Computing: Informatics and Systems* 32 (2021), p. 100603 (cit. on pp. 125, 130, 132, 141, 146).
- [11] María Gutiérrez et al. “Green-IN Machine Learning at a Glance”. In: *Computer* 56.6 (2023), pp. 35–43 (cit. on pp. 125, 130, 148).
- [12] Achim Guldner et al. “Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—Green Software Measurement Model (GSMM)”. In: *Future Generation Computer Systems* (2024) (cit. on pp. 125, 130).
- [13] Eleonora Fanouraki. *Did you know that 60% of game developers use game engines?* <https://www.slashdata.co/blog/did-you-know-that-60-of-game-developers-use-game-engines>. [Online; accessed 05-December-2023]. 2022 (cit. on pp. 127, 145).
- [14] Epic Games Inc. *Real-time round-up: the state of interactive 3D*. <https://www.unrealengine.com/en-US/blog/real-time-round-up-the-state-of-interactive-3d>. [Online; accessed 11-December-2023]. 2023 (cit. on p. 127).
- [15] activeplayer.io Game Statistics Authority. *Fortnite Live Player Count and Statistics*. <https://activeplayer.io/fortnite/>. [Online; accessed 11-December-2023]. 2023 (cit. on p. 127).

- [16] Epic Games Inc. *About Epic Games*. <https://www.epicgames.com/site/en-US/about>. [Online; accessed 13-March-2024]. 2024 (cit. on p. 127).
- [17] SlashData ©. *State of the Developer Nation 23rd Edition*. https://docsend.com/view/4cmnvxa2xb5jd6hr?utm_source=SlashData&utm_medium=FreeReports_SoN23. [Online; accessed 29-March-2024]. 2022 (cit. on p. 128).
- [18] Epic Games Inc. *Blueprint Compiler Overview*. https://dev.epicgames.com/documentation/en-us/unreal-engine/compiler-overview-for-blueprints-visual-scripting-in-unreal-engine?application_version=5.3. [Online; accessed 25-March-2024]. 2024 (cit. on pp. 129, 144).
- [19] University of Lille Inria. *PowerAPI*. <https://powerapi.org/>. [Online; accessed 27-May-2024]. 2023 (cit. on p. 130).
- [20] Nupur Kothari and Arka Bhattacharya. “JouleMeter: Virtual machine power measurement and management”. In: *MSR Tech Report* (2009) (cit. on p. 130).
- [21] CD PROJEKT Group. *Consolidated Report for Q1 2023*. <https://www.cdprojekt.com/en/investors/regulatory-announcements/consolidated-report-for-q1-2023/>. [Online; accessed 30-March-2024]. 2023 (cit. on p. 135).
- [22] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 146, 147).
- [23] Anders Andrae. *Projecting the chiaroscuro of the electricity use of communication and computing from 2018 to 2030*. Feb. 2019. DOI: 10.13140/RG.2.2.25103.02724 (cit. on p. 148).
- [24] Ankita Atrey, Nikita Jain, and N Ch Sriman Narayana Iyenger. “A Study on Green Cloud Computing”. In: *International Journal of Grid and Distributed Computing* 6 (Dec. 2013), pp. 93–102. DOI: 10.14257/ijgdc.2013.6.6.08 (cit. on p. 148).

- [25] Clauriton Siebra et al. “Towards a green mobile development and certification”. In: *2011 IEEE 7th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2011, pp. 288–294. DOI: 10.1109/WiMOB.2011.6085386 (cit. on p. 148).
- [26] R. Pereira et al. “Ranking programming languages by energy efficiency”. English. In: *Science of Computer Programming* 205 (2021). ISSN: 01676423. DOI: 10.1016/j.scico.2021.102609 (cit. on p. 148).
- [27] Lizhe Wang and Samee Khan. “Review of performance metrics for green data centers: A taxonomy study”. In: *The Journal of Supercomputing* 63 (Mar. 2011), pp. 1–18. DOI: 10.1007/s11227-011-0704-3 (cit. on p. 148).
- [28] Daniel Johnson et al. “Gamification and serious games within the domain of domestic energy consumption: A systematic review”. In: *Renewable and Sustainable Energy Reviews* 73 (2017), pp. 249–264 (cit. on p. 148).
- [29] Carlos Pérez et al. “A survey on green computing in video games: The dawn of Green Video Games”. In: (Dec. 2023). arXiv: 2312.09053 [cs.SE] (cit. on p. 149).
- [30] Carlos Pérez et al. “A comparative analysis of energy consumption between the widespread unreal and Unity video game engines”. In: (Feb. 2024). arXiv: 2402.06346 [cs.SE] (cit. on p. 149).
- [31] Jörg Kienzle et al. “Toward model-driven sustainability evaluation”. In: *Communications of the ACM* 63.3 (2020), pp. 80–91 (cit. on p. 150).
- [32] Gabriele Gramelsberger et al. “Enabling informed sustainability decisions: sustainability assessment in iterative system modeling”. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2023, pp. 964–968 (cit. on p. 150).
- [33] Istvan David and Dominik Bork. “Towards a taxonomy of digital twin evolution for technical sustainability”. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2023, pp. 934–938 (cit. on p. 150).

Part III

Discussion

Discussion

This part of the thesis discusses the results of this work by connecting the Research Questions posed in Part I with the results shown in the research articles that conform Chapters 1, 2, and 3 of Part II.

Research Questions

In this section of the thesis, the results of the three studies presented in the compendium are discussed. This discussion helps us to answer the Research Questions posed in Part I and thus aims to provide research directions in GSE research.

Figure 8 summarizes the answers to both Research Questions of this thesis indicating which elements of the research paper that is included in the compendium have contributed to the thesis.

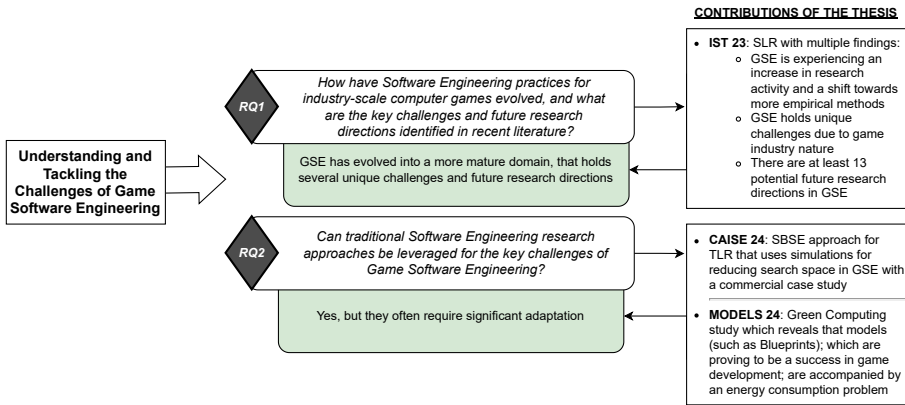


Figure 8: Research Questions answers overview and outcomes

Below, both the contribution and the answers will be detailed and explained further.

RQ₁: *How have Software Engineering practices for industry-scale computer games evolved, and what are the key challenges and future research directions identified in recent literature?*

The first step for answering this first question is to understand the current status of research on GSE. By analyzing the result of the SLR, included in the Chapter 1 of the Compendium (**IST 23** [1]), it is possible to observe that:

-
- Software Engineering practices for industry-scale computer games have gained significant momentum and maturity over the past decades.
 - Game Software Engineering faces several specific key challenges that differ from traditional Software Engineering ones.
 - Game Software Engineering offers many potential future research directions in which there is a lack of research.

Evolution of practices

One of the most striking indicators of the field's evolution is the substantial increase in research intensity. Comparing the period before 2009 with the period from 2009 to 2021, the SLR reveal a dramatic surge, with over four times more industry-scale research conducted in the latter period. This growth reflects the increasing recognition of video games as complex software systems that demand specialized engineering approaches.

Accompanying this increase in volume is a notable shift in research topics, indicating the maturation of the field. While earlier research (before 2009) tended to focus heavily on topics typical of a developing domain, such as Requirements or Specifications, the research landscape after 2009 is much more broadly distributed. Areas that have seen the biggest increase in interest and are now among the most studied include Design Tools and Techniques, Metrics, Software Architecture, Testing and Debugging, and Management. This shift towards topics like architecture, design, and metrics reflects the challenges presented by the increasing size and complexity of modern video games. It is noted, however, that the 2012 version of the ACM Computing Classification System (CCS) does not effectively represent the GSE domain, grouping many distinct areas of interest, such as Requirements/Specifications, Testing/Debugging, and Metrics, into a single category (5.3.4 Software verification and validation), suggesting a divergence of GSE from a traditional SE classification.

The research approaches employed in GSE have also evolved significantly. Exploratory studies, common in young research areas, have seen a drastic decrease, falling from over 40% before 2009 to a mere 1% in the subse-

quent period. At the same time, empirical approaches have more than doubled in interest. This rise in empirical research is seen as a sign of the field's maturation and a reflection of increased collaboration between industry and academia. Looking specifically at empirical methods, there has been another clear shift in preference. Before 2009, surveys were the most common empirical method. Since 2009, however, case studies have surged to become the dominant empirical method, accounting for over 50% of empirical studies. This transition from surveys to more in-depth case studies is interpreted as evidence of a narrowing industry-academia gap, with industry practitioners providing richer, real-world scenarios for researchers to study. It is also worth mentioning that, despite the increase in empirical methods, experiments have not shown a similar rise, which may be attributable to the fast-paced nature of the video game industry, where time-consuming experiments might be less feasible than case studies with more immediate impact.

While there is a substantial presence of industry-scale demonstrations in the research, the rigor in study reporting needs improvement, particularly concerning the critical examination of the work, including potential biases and limitations. Only a small number of studies were found to use best practices for reporting limitations.

Key challenges

The unique nature of video games and the dynamics of the industry present several key challenges that differentiate GSE from traditional SE:

- **Multidisciplinary Teams:** Game development requires integrating roles beyond engineering, such as artists, musicians, and designers, making communication and collaboration challenging.
- **Iterative and Evolving Design:** The design is often fluid and evolves alongside implementation, which is a different paradigm from more rigid SE methodologies.
- **Fixed Deadlines and Market Pressure:** Strict, immovable deadlines are common, with significant business consequences for delays.

-
- **Asset Management:** Modern games demand vast amounts of content and assets, requiring sophisticated management.
 - **Player Experience and Game-Specific Testing:** Beyond technical correctness, testing must evaluate subjective factors like "fun," "aesthetics," and "balance," requiring expertise beyond typical software testing.
 - **Subjective Non-functional Requirements:** Requirements are often abstract and user-centric, focusing on subjective experiences rather than concrete, measurable criteria.
 - **Complexity of Systems:** Games involve complex, interconnected systems like rendering, physics, sound, and AI that must be managed and synchronized effectively.
 - **Game Engine Expertise:** Heavy reliance on game engines necessitates specific skills, often involving scripting or domain-specific languages distinct from traditional programming.
 - **Generalization of Findings:** The inherent diversity in team sizes, platforms, genres, and underlying software makes it difficult to generalize research findings across the board.
 - **Adaptation of Methodologies:** While agile methods are adopted, they are often not implemented effectively or are insufficient on their own, contributing to issues like "crunch time".
 - **Testing Automation Difficulty:** The large codebases, high interactivity, asset volume, difficulty of reproducing bugs, and subjective nature of gameplay make test automation particularly challenging.
 - **Documentation Challenges:** Documenting the intertwined elements of software, art, and design is difficult, and current methods like Game Design Documents (GDDs) lack standardization.
 - **Games as a Service (GaaS):** The shift towards GaaS models increases the focus on software maintenance, presenting new challenges.

-
- **Security:** Multiplayer online games require very low-latency network systems that are at the same time robust and secure to handle traffic while combating cheating.

Future research directions

Thirteen potential future research directions for GSE are highlighted as areas ripe for further investigation:

1. **Cross-fertilization between GSE and MDE:** Investigating how Model Driven Engineering (MDE) can be applied in GSE and how GSE's use of models can also be useful for applying it in traditional Software Engineering MDE.
2. **Game software product lines:** Applying software product line techniques to manage reuse across games and platforms.
3. **The blossoming of publishing platforms:** Developing GSE techniques and tools specifically for the complex task of porting games across numerous diverse platforms.
4. **The multifaceted nature of GSE:** Research focused on developing approaches to address the challenge of generalizing findings given the heterogeneity of the GSE domain.
5. **Planning and scheduling issues:** Finding new best practices and adapting agile methods to different team sizes to mitigate or eliminate "crunch time".
6. **The consequences of Games as a Service:** Increased research on software maintenance practices due to the prevalence of the GaaS model.
7. **GSE security:** Focusing on building reliable, robust, and secure network systems for online multiplayer games.
8. **Weaving software and art design:** Improving documentation methods, possibly moving beyond traditional GDDs, to better capture the integrated nature of game development.

-
9. **A new wave of requirements research for GSE:** Applying Requirements Engineering (RE) to the unique, subjective requirements of games, such as user experience and gameplay balance.
 10. **Testing and debugging in the face of gameplay subjectivity:** Developing new techniques and automation for both traditional software testing and subjective gameplay testing.
 11. **Green GSE:** Research into improving code performance to reduce energy consumption in video games, an increasingly relevant ecological concern.
 12. **Search-based GSE:** Exploring the application of Search-Based Software Engineering (SBSE) techniques to GSE problems, potentially leveraging simulations or player data.
 13. **Content creation and artificial intelligence:** Analyzing the state of GSE research within the related domains of content creation and AI for video games.

RQ₁ answer: Game Software Engineering has evolved into a more mature and rapidly growing scientific domain, marked by a significant increase in research activity and a shift towards more empirical methods like case studies.

This field holds several unique challenges stemming from the multidisciplinary nature of game development, the complexity of systems and subjective requirements, and industry pressures.

Numerous future research directions have been identified, including leveraging game-specific assets like simulations for tasks like traceability link recovery, and addressing the environmental impact of game development by investigating energy efficiency across different implementation approaches.

RQ₂: *Can traditional Software Engineering research approaches be leveraged for the key challenges of Game Software Engineering?*

While the fundamental tasks and principles of traditional SE are often relevant to GSE, direct application of traditional methodologies may not

always be effective, and significant adaptation is frequently required to address the unique characteristics of video games.

Traceability Link Recovery (TLR) is an essential task in both classic software engineering and GSE. **CAISE 24** [2] presents an approach for supporting TLR in GSE that leverages video game simulations, specifically the reproduction of actions of Non-Player Characters (NPCs). Specifically, this approach uses NPC simulations to reduce the search space for tracing requirements to code. Afterwards, the approach incorporates traditional SE elements like Search-Based Software Engineering (SBSE) and Topic Modeling, although the core novelty lies in exploiting a game-specific asset (NPC simulations) which are common and typically created as part of video game development. The results show that this simulation-based approach significantly outperforms a baseline that does not incorporate simulations. While a fundamental SE task (TLR) is addressed, the method developed is tailored to the unique environment of video games, leveraging game-specific data and processes. This adaptation shows that traditional SE tasks can be addressed, but the methods may need to be adapted for leveraging them in the game domain.

MODELS 24 [3] focuses on comparing the energy consumption of visual scripting models, specifically Blueprints in Unreal Engine 5 (UE5), with traditional C++ coding for implementing game logic. This work exemplifies how traditional SE research approaches, such as conducting comparative empirical studies and using validated measurement frameworks, can be leveraged to investigating critical issues in GSE. However, there is a crucial difference in how energy consumption is measured and interpreted when moving from traditional software to video game software. For traditional software, energy consumption might often be measured simply as total energy over time (e.g., kWh). While the study initially reports power required over a fixed 30-second execution time, this metric alone is insufficient for video games. Video games are real-time applications where the frame rate or frames per second (FPS) impact on user experience and is even a requirement for publishing games on certain platforms. Therefore, to gain a better understanding of energy consumption in this context, it becomes essential to consider the energy consumed per average frame. This requires measuring not just the power drawn by

the system over time, but also the time it takes to render each frame. By combining these measurements, the power consumption per frame can be calculated.

This difference in measurement highlights again an adaptation needed when applying SE research to GSE. While the methodology of conducting empirical measurements in a controlled environment using validated tools aligns with traditional SE research, the metrics used to analyze the results must be tailored to the specific characteristics of the game domain.

The results presented demonstrate the importance of this adapted approach. While power measured over 30 seconds showed minimal differences between Blueprint and C++ implementations for the components studied, analyzing power required per average frame revealed noticeable distinctions. For the health system and inventory operations components, Blueprint models consumed more energy per frame than C++, with the difference being particularly significant for inventory operations (nearly 50% less energy per frame with C++). This contrast underscores why game-specific metrics like FPS are crucial for understanding the true energy efficiency implications within GSE.

RQ₂ answer: Traditional Software Engineering research approaches are certainly relevant to the key challenges of Game Software Engineering, but they often require significant adaptation to suit the unique characteristics of video game development. While direct application of traditional SE approaches or frameworks may not always be effective, SE tasks like TLR and analysis of energy consumption are applicable and find fertile ground in games when adapted.

Future work

A common factor in all three works included in the compendium is the need to continue investing efforts in GSE research. The increasing demands for performance and complexity in video games pose new challenges and it is essential to develop tools and methodologies that allow developers to create engaging games.

The use of NPC simulations has proven to be a promising approach for TLR, but it requires further exploration. It is essential to replicate studies across different video game domains to confirm their generalization. In addition, other maintenance tasks, such as bug localization, could benefit from the possibility of applying NPC simulations. In addition, NPC simulations have proven to be useful for reducing the search space, so simulations might also be meaningful for Search-Based Software Engineering.

Green Computing emerges as a critical area for GSE. Expanding energy consumption analysis to other platforms and development practices taking into account consumption per frame is necessary, considering the overall impact that video games can reach. Developing tools and better documentation that allow developers to monitor and optimize energy consumption during the development process would facilitate the creation of more sustainable games.

In addition, it is possible to report several challenges and opportunities for future research in both traditional Software Engineering and GSE:

- The success of models, templates, and visual tools such as Unity and Unreal Engine in video game development offers a learning opportunity for the broader **MDE** community. Bridging these fields could enhance both domains, making development more accessible to non-programmers and advancing software modeling techniques.
- Generally, systematic approaches like **software product lines** are underutilized in video games. The adaptation and optimization of reuse techniques for video game development is crucial, particularly for cross-platform compatibility and efficient porting.

-
- The diversity in team sizes, genres, and platforms makes generalizing findings across GSE projects challenging. Research should explore methodologies to create more adaptable and **universal solutions** for the varied needs of game development.
 - Current agile practices in GSE often lead to crunch time, stressing teams and reducing efficiency. Future studies should aim to **adapt agile methods** specifically for game development, addressing team diversity and the industry's unique demands to mitigate crunch culture.
 - The shift of part of the industry towards continuous development models like **Games as a Service** (GaaS) demands new approaches to maintenance, distribution, and enhancement. Research should focus on aligning these practices with the evolving industry requirements.
 - Multiplayer games require robust, **secure network systems** to handle synchronization, high traffic, and anti-cheat measures. Advances in GSE security are crucial to meet the growing demand for reliable online experiences.
 - The integration of software, art, and design in game development highlights the need for **better documentation practices**, such as dynamic Game Design Documents (GDDs), to support clear communication in multidisciplinary teams.
 - GSE requires tailored **requirements engineering** approaches that prioritize player enjoyment and gameplay balance. There is a need for tools and techniques that define and measure at least some of the subjective aspects of game quality.

As it has just been rendered, GSE is still a growing field with many areas yet to explore. However, analyzing the evolution of the video game market and the impact of GSE research on it will help focus efforts on the most relevant areas, ensuring that GSE effectively responds to the demands of industry.

Bibliography

- [1] Jorge Chueca et al. “The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”. In: *Information and Software Technology* (2023) (cit. on p. 161).
- [2] Javier Verón et al. “Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2024, pp. 352–367 (cit. on p. 167).
- [3] Javier Verón et al. “A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD”. In: *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 2024, pp. 114–125 (cit. on p. 167).

Part IV

Conclusions

Conclusions

This is the last part of the thesis, where the objective, results and achieved impact during the development of the research work is described.

The research field of Game Software Engineering (GSE) has experienced significant growth in recent years, as evidenced by the increase in the number of studies dedicated to this topic. This growth is also a reflection of the current growth of the video game industry, which demands new techniques and approaches to manage increasingly complex software.

The research developed in this thesis has produced results that have been published as three articles in specialized journals and relevant conferences with positive outcomes. After analyzing the results that are obtained in this work, the conclusions are:

The first work included in the compendium [1], in Chapter 1, presents a Systematic Literature Review (SLR) which reveals this growth in GSE research over the last decade. However, despite this progress, the field still lacks consolidation in terms of publication venues. There is a dispersion of studies across various journals and conferences not focused in the area of GSE, which hinders the construction of a cohesive body of knowledge. It is necessary to promote the creation of dedicated spaces for GSE research to foster greater collaboration and synergy among researchers.

The second work included in the compendium [2], in Chapter 2, focuses on TLR, a crucial process for software development and maintenance. This research work explores the novel application of Non-Player Character (NPC) simulations to optimize TLR in the context of video games. The results are promising, showing a significant reduction in the search space for TLR, which increases efficiency and accuracy in identifying traceability links. This approach, evaluated in an industrial video game, and validated by professional software engineers, opens new possibilities for improving requirements management in video game development.

The third work included in the compendium [3], in Chapter 3, addresses Green Computing in video game development, an emerging and vital area. Video games are played by so many people that small improvements in energy consumption could mean huge total savings in global energy consumption. This research work focuses on analyzing energy consumption in different implementation approaches for logic in Unreal Engine 5. It is observed that the use of code and software models present significant differences in terms of energy consumption for some logical

components implementation. This finding is relevant to the developer community as it suggests that an important factor to choose between both approaches should be the energy impact of each. However, further research is needed to better understand the energy implications of other components and development practices in video games.

This thesis presents a comprehensive view of GSE, exploring TLR and Green Computing for developing video game software. Each work in the compendium not only provides relevant findings for the scientific community, but also offers valuable challenges and future research directions to explore in GSE. Thus, researchers and professional video game developers can use the outcomes of this work as starting point to improve GSE.

The proliferation of studies in recent years is encouraging; however, their dispersion across various journals and conferences hinders the construction of a solid and accessible body of knowledge. The creation of dedicated spaces for publishing GSE research should be promoted, as well as fostering collaboration among researchers. Improving rigor in documentation and study reporting is also crucial to facilitate comparison and replicability, so GSE is lead to meaningful and informed decision-making in both academia and industry.

Bibliography

- [1] Jorge Chueca et al. “The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games”. In: *Information and Software Technology* (2023) (cit. on p. 177).
- [2] Javier Verón et al. “Improving Requirement Traceability by Leveraging Video Game Simulations in Search-Based Software Engineering”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2024, pp. 352–367 (cit. on p. 177).
- [3] Javier Verón et al. “A Comparative Analysis of Energy Consumption Between Visual Scripting models and C++ in Unreal Engine: Raising Awareness on the importance of Green MDD”. In: *Proceedings of the ACM/IEEE*

27th International Conference on Model Driven Engineering Languages and Systems. 2024, pp. 114–125 (cit. on p. 177).