# Are Game Engines Software Frameworks? A Three-perspective Study*

Cristiano Politowski[a,*], Fabio Petrillo[b], João Eduardo Montandon[c], Marco Tulio Valente[c] and Yann-Gaël Guéhéneuc[a]

[a]Concordia University, Montreal, Quebec, Canada
[b]Université du Québec à Chicoutimi, Chicoutimi, Quebec, Canada
[c]Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

ARTICLE INFO

*Keywords*:
Game-engine
Framework
Video-game
Mining
Open-source

Abstract

Game engines help developers create video games and avoid duplication of code and effort, like frameworks for traditional software systems. In this paper, we explore open-source game engines along three perspectives: literature, code, and human. First, we explore and summarise the academic literature on game engines. Second, we compare the characteristics of the 282 most popular engines and the 282 most popular frameworks in GitHub. Finally, we survey 124 engine developers about their experience with the development of their engines. We report that: (1) Game engines are not well-studied in software-engineering research with few studies having engines as object of research. (2) Open-source game engines are slightly larger in terms of size and complexity and less popular and engaging than traditional frameworks. Their programming languages differ greatly from frameworks. Engine projects have shorter histories with less releases. (3) Developers perceive game engines as different from traditional frameworks. Generally, they build game engines to (a) better control the environment and source code, (b) learn about game engines, and (c) develop specific games. We conclude that open-source game engines have differences compared to traditional open-source frameworks although this differences do not demand special treatments.

## 1. Introduction

> "*It's hard enough to make a game (...). It's really hard to make a game where you have to fight your own tool set all the time.*"
> — Schreier [1] quoting a game developer on the difficulties faced using their game engine.

For decades, video games have been a joyful hobby for many people around the world [2], making the game industry multi-billionaire, surpassing the movie and music industries combined [3]. However, realistic graphics and smooth gameplays hide constant and non-ending problems with game development, mostly related to poor software-development practices and inadequate management [4]. Problems like these result in a scenario where 80% of the top 50 games on Steam[1] need critical updates [5], also leaving a trail of burnout developers after long periods of "crunchs"[2] [6].

During game development, developers use specialized software infrastructures to develop their games; chief among which are *game engines*. Game engines encompass a myriad of resources and tools [7–10]. They can be built from scratch during game development, reused from previous games, extended from open-source ones, or bought off the shelves. They are essential to game development but misunderstood and misrepresented by the media [11] and developers due to lacks of clear definitions, architectural references [12], and academic studies. They are also the source of problems, especially between design and technical teams [13, 14].

To address these problems, some researchers suggest the use of software-engineering techniques [4, 15, 16] while others consider game development as a special kind of software and propose new engineering practices or extensions to classical ones [17–23]. However, they did not study a large number of game engines, either proprietary, because only 13% of all the games on Steam describe their engines [24], or open source. They also did not survey game engine developers.

Therefore, we set to comparing open-source video-game engines with traditional open-source software frameworks can help researchers and developers to understand them better. With this article, we want to answer whether game engines share similar characteristics with software frameworks. By comparing the tools (engines and frameworks) rather than their instances (video games, traditional software systems), we provide a distinct view on game development: rather than studying how developers use games engines, we focus on how the foundations of their games are built.

We study open-source game engines from three perspectives: *literature*, *code*, and *human* to provide an global view on the states of the art and practice on game engines. We explore academic and gray literature on game engines; compare the characteristics of the 282 most popular engines and the 282 most popular frameworks in GitHub; and, survey 124 engine developers about their experience with the development of their engines. Thus, we provide four contributions: (1) a corpus of open-source engines for this and

*Dataset: https://doi.org/10.5281/zenodo.3606899.

*Corresponding author
c_polito@encs.concordia.ca (C. Politowski); fabio@petrillo.com (F. Petrillo); joao.montandon@dcc.ufmg.br (J.E. Montandon); mtov@dcc.ufmg.br (M.T. Valente); yann-gael.gueheneuc@concordia.ca (Y. Guéhéneuc)
ORCID(s): 0000-0002-0206-1056 (C. Politowski); 0000-0002-8355-1494 (F. Petrillo); 0000-0002-3371-7353 (J.E. Montandon); 0000-0002-8180-7548 (M.T. Valente); 0000-0002-4361-2563 (Y. Guéhéneuc)

[1]Steam is a video game digital distribution service platform available at https://store.steampowered.com/.

[2]In game development, "crunch time" is the period during which developers work extra hours to deliver their game in time.

future research work; (2) an analysis and discussion of the characteristics of engines; (3) a comparison of these characteristics with those of traditional frameworks; and, (4) a survey of engine developers about their experience with engine development.

We show that, different from what researchers and engine developers think, there are qualitative but no quantitative differences between engines and frameworks. Game engines are slightly larger in terms of size and complexity and less popular and engaging than traditional frameworks. The programming languages of game engines differ greatly from that of traditional frameworks. game engine projects have shorter histories with less releases. Developers perceive game engines as different from traditional frameworks and claim that engines need special treatments. Developers build game engines to better control the environment and source code, to learn about game engines, and to develop specific games. We conclude that open-source game engines have differences compared to traditional open-source frameworks although this differences, related to community, do not demand special treatments.

The paper is structured as follows: Section 2 lists the research questions for all the three perspectives: literature, code, human. Section 3 shows the results of the literature perspective. Section 4 described the study design for the code perspective: metrics, data collection, and analysis. Section 6 discusses our results and threats to their validity. Section 7 concludes.

In addition, the detailed results are in the Appendix Sections. Appendix A shows results related to static analysis of the projects. Appendix B shows the historic analysis of the projects. Appendix C shows the community analysis of the projects. Appendix 5 describes our survey of engine developers for the human perspective.

## 2. Research Questions

This Section shows the list of research questions and the metrics used to answer them. An overview of the Perspectives, RQs, and Metrics, is in the Appendix Figure 6. More details about the metrics are into the Table 1.

### 2.1. RQ1: Literature Perspective

Although software frameworks are part of the toolset of most developers nowadays, its concept is often misunderstood, specially with libraries[3]. To better understand the differences between frameworks and game engines we explore the *literature* perspective using Scopus[4], for academic books and the *search engines* on internet, for articles, technical blogs, and discussion forums. In the Section 3 we aim to answer the following research questions.

- RQ1.1: What is the definition for *software framework*?
- RQ1.2: What is the definition for *game engine*?
- RQ1.3: What are the *works* related to game engines?

### 2.2. RQ2: Code Perspective

With respect to the design and implementation of game engines and traditional frameworks, we study their static, historical, and community characteristics.

#### *RQ2.1: Static Characteristics*

To understand the differences of the frameworks and game engines projects from a code perspective, we investigate the static attributes of the projects, like their size, complexity of the functions, programming languages and licenses used. In the Section 4.4 (and Appendix A) we aim to answer the following research questions.

- RQ2.1.1: What is the popularity of the *languages* in the projects?
  - Metrics: `main_language`
- RQ2.1.2: What is the popularity of the *licenses* in the projects?
  - Metrics: `license`
- RQ2.1.3: What are the *project sizes* of engines and frameworks?
  - Metrics: `main_language_size`, `total_size`, `n_files`
- RQ2.1.4: What are the *function sizes* of engines and frameworks?
  - Metrics: `n_funcs`, `nloc_mean`, `func_per_file_mean`
- RQ2.1.5: What are the *function complexities* of engines and frameworks?
  - Metrics: `cc_mean`

#### *RQ2.2: Historical Characteristics*

To explore the historical characteristics of the projects, we compare the life-cycles of game engines and traditional frameworks. We analyze the tags released (versions), projects' lifespan and commits. In the Section 4.5 (and Appendix B) we aim to answer the following research questions.

- RQ2.2.1: How many versions were *released* for each project?
  - Metrics: `tags_releases_count`
- RQ2.2.2: What is the *lifetime* of the projects?
  - Metrics: `lifespan`
- RQ2.2.3: How frequently do projects receive new *contributions*?
  - Metrics: `commits_count`, `commits_per_time`
- RQ2.2.4: Are commits made on game engines more *effort-prone*?
  - Metrics: `lines_added`, `lines_removed`, `code_churn`

[3] We encountered many problems while gathering the dataset as developers labeled their projects incorrectly; for example, labelling a library a "framework" or a game an "engine".

[4] https://www.scopus.com

***RQ2.3: Community Characteristics***

To investigate the interactions of the OSS community on the projects, we analyze the popularity of the projects, the number of issues reported in these projects, and the truck-factor measure [25]. In the Section 4.6 (and Appendix C) we aim to answer the following research questions.

- RQ2.3.1: How many developers *contribute* in the project?
  - Metrics: `truck_factor`
- RQ2.3.2: How *popular* are the projects considering their main languages?
  - Metrics: `stargazers_count`, `contributors_count`
- RQ2.3.3: How many *issues* are reported in each project?
  - Metrics: `issues_count`, `closed_issues_count`, `closed_issues_rate`

## 2.3. RQ3: Human Perspective

The human perspective pertains to the developers' perception of game engines and of their differences with traditional frameworks. We conducted an online survey with developers of the game engines to understand why they built such engines and their opinions about the differences (if any) between engines and frameworks.

Question 1 contains a predefined set of answers that we compiled from the literature and from the documentation and "readme" files studied during the manual filtering of the datasets. The respondent could choose one or more answers. We also provided a free-form text area for developers to provide a different answer and–or explain their answers. With Question 2, we want to understand whether game engine developers are also traditional software developers. Finally, Question 3 collected the developers' point of views regarding the differences (or lack thereof) between the development of engines and frameworks.

- RQ3.1: What are the reasons developers create open-source game engines?
  - Survey question 1: Why did you create or collaborated with a video-game engine project?
    - To help me to create a game
    - To learn how to build an engine
    - To have the full control of the environment
    - Because the existent engines do not provide the features I need
    - Because I wanted to work with this specific programming language
    - Because the licenses of the existent engines are too expensive
    - Other [please specify]
- RQ3.2: Do game engine developers also have expertise with traditional software?
  - Survey question 2: Have you ever written code for a software unrelated to games, like a Web, phone, or desktop app? [Yes or No]
- RQ3.3: For game engine developers, is it similar to developing a traditional framework?
  - Survey question 3:How similar do you think writing a video-game engine is compared to writing a framework for traditional apps? (Like Django, Rails, or Vue) – [1 (very different) to 5 (very similar)]

# 3. Results from RQ1: Literature Perspective

We study game engines along the literature perspective by querying both Scopus and the Internet. We report that only few works on game engines exist: mostly books, few academic papers. We did not perform a systematic literature review (SLR) [26] because of the small size of the current academic literature on the topic, as shown in the following.

## RQ1.1: What is the definition for *software framework*?

GitHub uses a set of "topics"[5] to classify projects. It defines the topic "framework" as *"a reusable set of libraries or classes in software. In an effort to help developers focus their work on higher level tasks, a framework provides a functional solution for lower level elements of coding. While a framework might add more code than is necessary, they also provide a reusable pattern to speed up development."*

Pree [27] defined frameworks as having *frozen* and *hot* spots: code blocks that remain unchanged and others that receive user code to build the product. Larman [28] observed that frameworks use the *Hollywood Principle*, "Don't call us, we'll call you.": user code is called by the framework. Taylor [29] sees a framework as a programmatic bridge between concepts (such as "window" or "image") and lower-level implementations. Frameworks can map architectural styles into implementation and–or provide a foundation for an architecture.

## RQ1.2: What is the definition for *game engine*?

ID Software[6] introduced the concept of video-game engine in 1993 to refer to the technology "behind the game" when they announced the game DOOM [7, 30]. In fact, they invented the game engine around 1991 and revealed the concept around the DOOM press release early 1993 [31].

The invention of this game technology was a discrete historical event in the early 1990s but it established MS-DOS 3.3 as a relevant gaming platform, mostly because of the NES-like horizontal scrolling emulation, allowing developers to create games similar to the ones on Nintendo console. It also introduced the separation of game engine from "assets" accessible to players and thereby revealed a new paradigm for game design on the PC platform [31], allowing

[5] https://github.com/topics/framework
[6] https://www.idsoftware.com

players to modify their games and create new experiences. This concept has since evolved into the "fundamental software components of a computer game", comprising its core functions, e.g., graphics rendering, audio, physics, AI [30].

John Carmack[7], and to a less degree John Romero[8], are credited for the creation and adoption of the term game engine. In the early 90s, they created the first game engine to *separate the concerns* between the game code and its assets and to *work collaboratively* on the game as a team [14, 30]. Also, they "lent" their engines to other game companies to allow other developers to focus only on game design.

In 2002, Lewis and Jacobson [32] defined game engines as "*collection[s] of modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data)*". In 2007, Sherrod [10] defined engines as frameworks comprised of different tools, utilities, and interfaces that hide the low-level details of the implementations of games. Engines are extensible software that can be used as the foundations for many different games without major changes [7] and are *"software frameworks for game development"*. They relieve developers so that they can focus on other aspects of game development"[9]. In 2019, Toftedahl and Engström [24] analysed and divided engines in four complementary types: (a) *Core Game Engine*, (b) *Game Engine*, (c) *General Purpose Game Engine*, and (d) *Special Purpose Game Engine*.

### RQ1.3: What are the *works* related to game engines?

There are few academic papers on game engines. Most recently and most complete, Toftedahl and Engström [24] analysed the engines of games on the Steam and Itch.io platforms to create a taxonomy of game engines. They highlighted the lack of information regarding the engines used in mainstream games with only 13% of all games reporting information about their engines. On Steam, they reported Unreal (25.6%), Unity (13.2%), and Source (4%) as the main engines. On Itch.io, they observed that Unity alone has 47.3% of adoption among independent developers.

Messaoudi et al. [12] investigated the performance of the Unity engine in depth and reported issues with CPU and GPU consumption and modules related to rendering.

Cowan and Kapralos [33] in 2014 and 2016 [34] analysed the game engines used for the development of serious games. They identified few academic sources about tools used to develop serious games. They showed that "Second Life"[10] is the most mentioned game engine for serious games, followed by Unity and Unreal. They considered game engines as parts of larger infrastructures, which they call frameworks and which contain scripting modules, assets, level editors as well as the engines responsible for sound, graphics, physics, and networking. They ranked Unity, Flash, Second Life, Unreal, and XNA as the most used engines.

Neto and Brega [35] conducted a systematic literature review of game engines in the context of immersive applications for multi-projection systems, aiming at proposing a generic game engine for this purpose.

Wang and Nordmark [36] assumed that game development is different from traditional software development and investigated how architecture influences the creative process. They reported that the game genre significantly influences the choice of an engine. They also showed that game-engine development is driven by the creative team, which request features to the development team until the game is completed. They observed that adding scripting capability ease game-engine development through testing and prototyping.

Anderson et al. [37] raised issues and questions regarding game engines, among which the need for a unified language of game development, the identification of software components within games, the definition of clear boundaries between game engines and games, the links between game genres and game engines, the creation of best practices for the development of game engines.

**Summary for RQ1: Literature Perspective**

We could not find many academic paper on game engines or a reliable source for gray literature. We recommend to extend this work with multivocal literature review (with academic and grey literature).

## 4. Results from RQ2: Code Perspective

This section details the method for gathering the data and the metrics used to answer the set of RQs 2.1, 2.2, and 2.3. It also introduced the applied statistical techniques. For the sake of clarity, this section does not provide all the details but summarises the answers to each set of RQs. The Appendixes A, B, and C present the detailed results of each set of RQs.

### 4.1. Method

Figure 1 shows the steps that we followed to mine the data to answer our questions. In Step 1, on August 8, 2019, we gathered the top 1,000 projects in GitHub related to the *game-engine* and *framework* topics, separately, storing each one in a specific dataset.

In Step 2, we filtered these projects using the following criteria to remove "noise" and improve the quality of our dataset, which is a common approach when dealing with Github repositories [25, 38] to obtain 458 engines and 743 frameworks:

- The project must have *more than one contributor*;
- The project must have been *starred at least twice*;
- The *last commit* must be at least from *2017*;
- The project *cannot be archived*.

[7] John Carmack was the lead programmer and co-founder of id Software in 1991.

[8] John Romero was the designer, programmer and also co-founder of id Software in 1991.

[9] https://github.com/topics/game-engine

[10] Second Life is not a game engine per se but a game that can be extended by adding new "things" through "mod" or "modding".

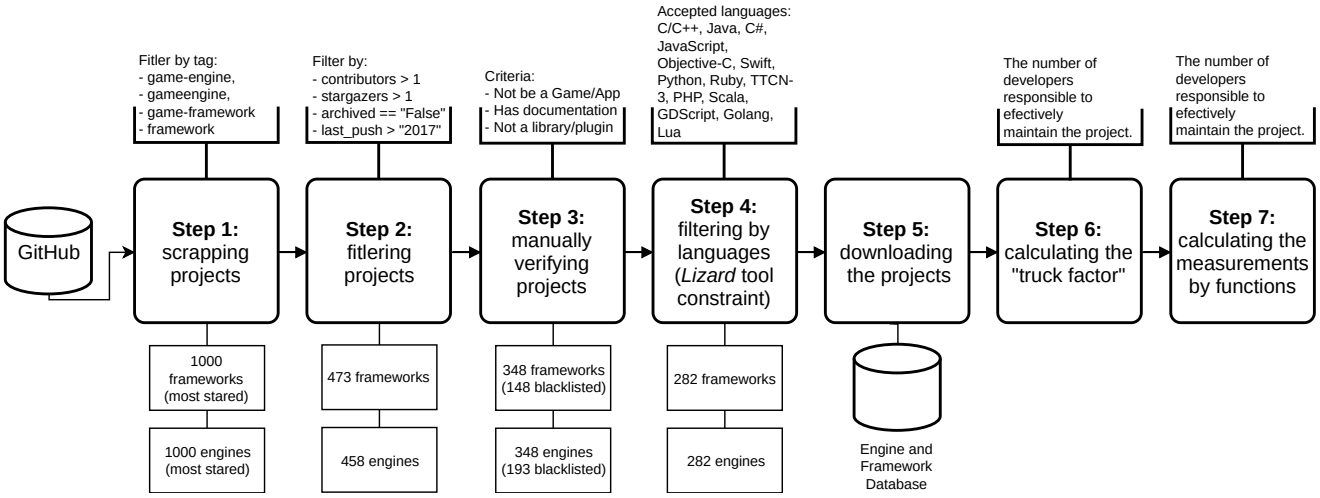



**Figure 1:** The process used to gather the data from GitHub projects.

In Step 3, we manually analysed the remaining 458 + 743 = 1,201 projects to remove those that are neither game engines nor frameworks according to the definitions in Section 3. We kept 358 game engines and 358 frameworks.

In Step 4, we kept only projects with programming language supported by Lizard: C/C++, C#, GDScript, Golang, Java, JavaScript, Lua, Objective-C, PHP, Python, Ruby, Scala, Swift, TTCN-3. We had now 282 engines and 282 frameworks.

In Step 5, we computed the metrics and stored their values in the datasets, which we describe in details in the following Section 4.2.

In Step 6, we computed the *truck-factor* of each project, which is the number of contributors that must quit before a project is in serious trouble [25, 39].

In Step 7, we used Lizard to gather the average value of the metrics related to functions. Lastly, we ordered the projects by popularity: how many "stars" they have.

Figure 2 shows an example containing the Github page of the engine Godot[11]. We only consider the main language of the projects but most projects are composed of multiple languages. Almost all the code of Godot is written in C++ (93%). Godot is tagged with the "game-engine" topic and, therefore, was found through our search. Godot is the most popular engine containing more than 26K votes.

The dataset, scripts and all the material from this study are in its replication package[12].

## 4.2. Metrics

Defining metrics is challenging. Some authors warn about problems with simplistic measurements [40] and lack of precision of tools that make the measurements [41]. However, imperfect quantification is better than none [42].

Kaner et al. [40] recommends the use of direct metrics[13] but also defines a framework to described and justify the metrics. We used a simplified version of this framework with six questions: *Purpose* (What is the purpose of this metric?), *Scope* (What is the scope of this metric?), *Scale* (What is the natural scale of the attribute we are trying to metric?), *Domain* (What is the domain this metric belongs to?), and *Instrument* (How the metric will be measured?).

Therefore, to answer the questions RQ2.1: Static Characteristics, RQ2.2: Historical Characteristics, and RQ2.3: Community Characteristics, we use the set of metrics described in Table 1. Also, the following list some details about some of the metrics.

*RQ2.1.1. Metric:* `main_language` – According to Tiobe index, currently, the most common languages are C, Java, Python[14]. Also, GitHub uses Linguistic to determine the most common language in the project[15].

*RQ2.1.2. Metric:* `license` – MIT, Apache-2.0, and GPL-V3 were the most common open source licenses in 2019[16]. GitHub has a "license picker" allowing the user to choose from different open-source licenses[17].

*RQ2.1.3. Metrics:* `main_language_size`, `total_size`, `n_file` – GitHub recommends repositories with less than 1GB, and less than 5GB is strongly recommended. Also with 100MB maximum file size limit[18].

*RQ2.1.4. Metrics:* `n_func`, `nloc_mean`, `func_per_file_mean` – Lizard[19] gives a list of all functions in the project with

[11] https://github.com/godotengine/godot

[12] https://doi.org/10.5281/zenodo.3606899.

[13] Direct metric is a metric that does not depend upon a measure of any other attribute.

[14] https://www.tiobe.com/tiobe-index/

[15] https://github.com/github/linguist

[16] https://bit.ly/3f4myu3

[17] https://bit.ly/32YNOYv

[18] https://bit.ly/2BDuKns

[19] https://github.com/terryyin/lizard

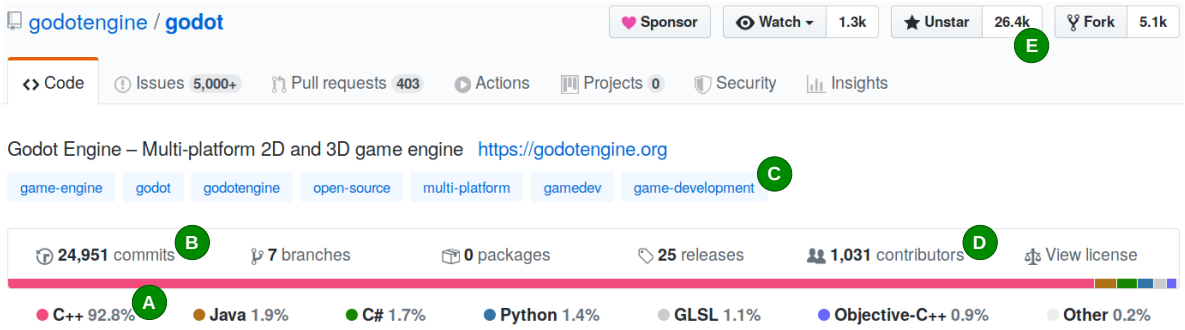



**Figure 2:** Godot engine Github page example. In this case we considered C++ as *main language* (A) and filtered the projects with least one commit from 2017 or newer (B) and with more than two contributors (D). *Tags*, provided by the developers, were used to search for the engines (C). Finally, we ordered the projects by the *stars* (E).

NLOC (lines of code without comments) and CCN (cyclomatic complexity number). It also gives the list of files and the functions' name (signatures).

*RQ2.1.5. Metrics:* `cc_mean` – McCabe's [43] recommends keeping the complexity of modules below 10.

*RQ2.2.1. Metric* `tags_releases_count`*:* Within the context of GitHub, a tag is a release of the product.

*RQ2.2.2. Metrics* `commits_count`, `commits_per_time` – PyDriller is a Python framework to analyze Git repositories[20].

*RQ2.2.4. Metrics:* `lines_added`, `lines_removed`, `code_churn` – Code churn measures the number the amount of code changes occurred during development of code [44]. We use the sum of deleted and removed lines.

*RQ2.3.1. Metric:* `truck_factor` – According to Avelino et al. [25], truck-factor is the number of people on the team that have to be hit by a truck before the project becomes problematic. Therefore, systems with low truck-factor have problems with strong dependency of certain developers. Linux has truck-factor of 57 and Git 12. We used a library defined by Avelino et al. [25] as instrument.

*RQ2.3.2. Metrics:* `stargazers_count` – "vuejs" is the most stared github opensource software project with more than 169K[21]. In 2019, "microsoft/vscode" had the highest number of contributors with 19.1K[22].

### 4.3. Analysis

We used the statistical-analysis workflow-model for empirical software-engineering research [45] to test statistically the differences between engines and frameworks. For each continuous variable, we used descriptive statistics in the form of tables with mean, median, min, and max values, together with boxplots. For the boxplots, to better show the distributions, we removed outliers using the standard coefficient of 1.5 ($Q3 + 1.5 \times IQR$). We observed outliers for all the measures, with medians skewed towards the upper quartile (Q3). To check for normality, we applied the Shapiro test [46] and checked visually using Q–Q plots. Normality $< 0.05$ means the data is not normally distributed. Finally, given the data distribution, we applied the appropriate statistical tests and computed their effect sizes.

### 4.4. Results for RQ2.1: Static Characteristics

Table 2 shows the results of Wilcoxon tests. The p-values $< 0.01$ indicate that the distributions are not equal and there is a significant difference between engines and frameworks, although this difference is *small*. The biggest effects are related to source code metrics, i.e., *nloc_mean* and *cc_mean*.

The implementation of game engines and traditional frameworks are different but without statistical significance. Engines are bigger and more complex than frameworks. They use mostly compiled programming languages vs. interpreted ones for frameworks. They both often use the MIT license.

### 4.5. Results for RQ2.2: Historical Characteristics

Overall, all metrics have similar median values when comparing both groups, except for *tags_releases_count*. In fact, engines releases way less versions (median is one) than frameworks (median is 32).

Table 3 shows the results of Wilcoxon tests, showing large differences for all historical measures except `lines_added`, `lines_removed`, and `code_churn`. Versioning does not look like a well-followed practice in engine development, with few versions compared to frameworks. Commits are less frequent and less numerous in engines, which are younger and have shorter lifetimes when compared to frameworks.

### 4.6. Results for RQ2.3: Community Characteristics

Table 4 shows the results of Wilcoxon tests, indicating a large difference in all measures related to community. The truck-factor shows that the majority of the projects have few contributors. Some uncommon languages, like Go and C#, are popular compared to others in more prevalent projects, e.g., C++ and JavaScript.

[20] https://github.com/ishepard/pydriller
[21] https://github.com/vuejs/vue
[22] https://octoverse.github.com/

**Table 1**
Description of the Metrics for the Code Perspective (Rqs 2, 3, and 4). We adapted the framework defined by Kaner et al. [40].

| RQs | Purpose | Scope | Metric | Scale | Domain | Instrument |
|---|---|---|---|---|---|---|
| RQ2.1.1 | Verify what is the most common main languages by the projects | Project | main_language | Nominal | Programming languages | API GraphQL (V4) |
| RQ2.1.2 | Verify what is the most common licenses used by the projects | Project | license | Nominal | Source code licenses | API GraphQL (V4) |
| RQ2.1.3 | Verify the project size | Project | main_language_size | Ratio | Positive rational numbers (Q) | API GraphQL (V4) |
| | | | total_size | Ratio | Positive rational numbers (Q) | |
| | | | n_file | Ratio | Natural numbers (N) | |
| RQ2.1.4 | Verify the function size | Function | n_func | Ratio | Natural numbers (N) | Lizard |
| | | | nloc_mean | Ratio | Positive rational numbers (Q) | |
| | | | func_per_file_mean | Ratio | Positive rational numbers (Q) | |
| RQ2.1.5 | Verify the complexity of the function | Function | cc_mean | Ratio | Natural numbers (N) | Lizard |
| RQ2.2.1 | Verify the release strategy of the project | Project | tags_releases_count | Ratio | Natural numbers (N) | API GraphQL (V4) |
| RQ2.2.2 | Verify the lifetime of the project | Project | lifespan | Ratio | Natural numbers (N) | API GraphQL (V4) |
| RQ2.2.3 | Verify the contributions to the project | Commits | commits_count | Ratio | Natural numbers (N) | Pydriller |
| | | | commits_per_time | Ratio | Positive rational numbers (Q) | |
| RQ2.2.4 | Very the effort made in the projects | Commits | lines_added | Ratio | Natural numbers (N) | Pydriller |
| | | | lines_removed | Ratio | Natural numbers (N) | |
| | | | code_churn | Ratio | Natural numbers (N) | |
| RQ2.3.1 | Verify the contribution of the project | Contributors | truck_factor | Ratio | Natural numbers (N) | Library Avelino et. al. |
| RQ2.3.2 | Verify the popularity of the project | Project | stargazers_count | Ratio | Natural numbers (N) | API GraphQL (V4) |
| | | | contributors_count | Ratio | Natural numbers (N) | |
| RQ2.3.3 | Verify the how developers deal with issues in the project | Issues | issues_count | Ratio | Natural numbers (N) | API GraphQL (V4) |
| | | | closed_issues_count | Ratio | Natural numbers (N) | |
| | | | closed_issues_rate | Ratio | Positive rational numbers (Q) | |

**Summary for RQ2: Code Perspective**

We observed that, although *static* characteristics of game engines and frameworks are similar, the *community* of these projects differ. Also, *historical* aspects are mixed, as engines have a smaller lifespan and less releases, yet similar effort on commits contribution and code churn.

## 5. Results from RQ3: Human Perspective

We now discuss developers' own perception of game engines and of their differences with traditional frameworks. We used an online form to contact developers over a period of three days. We sent e-mails to 400 developers of the game engines in our dataset, using the truck-factor of each project: developers who collaborate(d) most to the projects. We received 124 responses, i.e., 31% of the developers. The survey, answers, and scripts for their analyses are in the repli-

**Table 2**
Statistical Tests, RQ2.1: Static Characteristics

| Variable | P-value | Estimate | Effect |
|---|---|---|---|
| main_language_size | <0.01 | 0.28 | 0.189 (small) |
| total_size | <0.01 | 0.34 | 0.188 (small) |
| n_file | <0.01 | 45.00 | 0.155 (small) |
| n_func | <0.01 | 769.00 | 0.211 (small) |
| nloc_mean | <0.01 | 2.12 | 0.297 (small) |
| func_per_file_mean | <0.01 | 3.13 | 0.208 (small) |
| cc_mean | <0.01 | 0.53 | 0.356 (small) |

**Table 3**
Statistical Tests, RQ2.2: Historical Characteristics

| Variable | P-value | Estimate | Effect |
|---|---|---|---|
| tags_releases_count | <0.01 | -24.00 | -0.613 (large) |
| lifespan | <0.01 | -56.29 | -0.32 (large) |
| commits_count | <0.01 | -175.00 | -0.198 (large) |
| commits_per_time | <0.01 | -0.30 | -0.198 (large) |
| lines_added | <0.01 | 134.47 | 0.219 (small) |
| lines_removed | <0.01 | 48.83 | 0.168 (small) |
| cchurn_delta | <0.01 | 153.88 | 0.224 (small) |
| cchurn_sum | <0.01 | 222.48 | 0.212 (small) |

**Table 4**
Statistical Tests, RQ2.3: Community Characteristics

| Variable | P-value | Estimate | Effect |
|---|---|---|---|
| stargazers_count | <0.01 | -358.00 | -0.511 (large) |
| contributors_count | <0.01 | -9.00 | -0.459 (large) |
| truck_factor | 0.01 | <0.01 | -0.138 (large) |
| issues_count | -139.00 | <0.01 | -0.451 (large) |
| closed_issues_count | -122.00 | <0.01 | -0.459 (large) |
| closed_issues_rate | -0.05 | <0.01 | -0.27 (large) |

cation package[12].

## Question 1: Why did you create or collaborated with a video-game engine project?

Figure 3 shows the breakdown of the developers' answers. Having access to the source code, freedom to develop, etc., i.e., *control of the environment*, is the developers' major reason for working on a game engine while *learning* to build an engine is the second reason; explaining why many engines have few developers and commits.

The third reason is to build a *game*, confirming the lack of clear separation between developers and game designers. It is indeed common for game developers to act also as game designers, specially in independent games, e.g., the single developer of Stardew Valley[23].

The next answer is about working with a specific *language*, also related with learning: when learning a new language, developers want to apply or test their knowledge on some projects, and game engines are interesting candidates.

[23] https://www.stardewvalley.net/

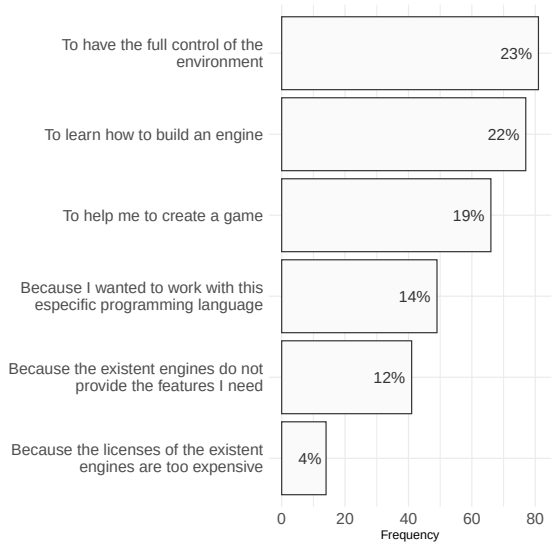



**Figure 3:** Answers to Question 1: Why did you create or collaborated with a video-game engine project?

Also related to the environment, the next answer concerns the *features* offered by existing engines: reusing or creating a new engine may be necessary for certain, particular games with specific requirements. Developers think as game designers: the game concept(s) may require a new engine.

The engine *licenses* are the least concern: fees and taxes from vendors, e.g., Unreal and Unity, are not important to developers because some licenses are "indie" friendly and offer low rates for indie games [24].

Finally, 19 developers provided "Other" answers: they work on game engines because "it is fun" and–or they have access to some source code, e.g., one developer who reverse-engineered a proprietary engine wrote:

> *"The source for the original engine was proprietary and so we opened the platform by reverse-engineering it then re-implementing under GPL3."*

Other answers include performance, platform compatibility, new experimental features, and creating a portfolio.

## Question 2: Have you ever written code for a software unrelated to games, like a Web, phone, or desktop app?

The great majority of developers, 119 of the 124 respondents (96%), have experience with traditional software. The respondents can be considered general software developers with expertise in engine development.

## Question 3: How similar do you think writing a video-game engine is compared to writing a framework for traditional apps? (Like Django, Rails, or Vue)

Figure 4 shows that engine developers consider engines different from frameworks: 59% of the respondents believe

that engines follows a different process from frameworks. Only 20% believe this this process is similar. This is a surprising result as they also have experience in developing traditional software.

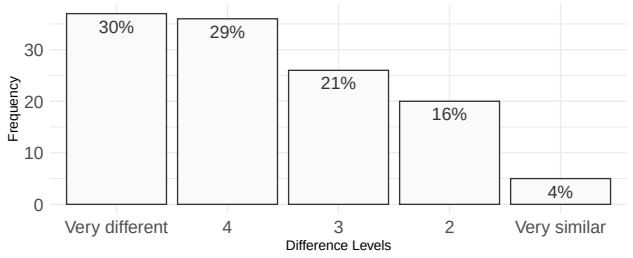



**Figure 4:** Answers to Question 3: How similar do you think writing a video-game engine is compared to writing a framework for traditional apps? (Like Django, Rails, or Vue)

**Summary for RQ3: Human Perspective**

The developers' main reasons to work on an engine is (a) having better control over the environment and source code, (b) learning game-engine development, and (c) helping develop a specific game. Almost all the engine developers have experience with traditional software. They consider these two types of software as different.

## 6. Discussions

We now discuss the results of our study of engines along the three perspectives.

### 6.1. Perils for Engines and Frameworks

Kalliamvakou et al. [47] analysed developers' usages of GitHub and reported a set of perils and promises related to GitHub projects. Table 5 shows the perils applying to the objects of our study: engines and frameworks. The perils 7, 8 (about pull-requests), and 9 (activity outside GitHub) are out of scope of our dataset.

In Peril 1, the authors distinguished forks and base repositories. In our search, we observed that most repositories are base ones. We found few forks that we removed during the manual filtering. Therefore, this peril is false for both engines and frameworks.

In Peril 2, the authors reported that the median of commits were 6, with 90% of the projects having less than 50 commits. We observed that the engines and frameworks in our dataset have medians of 616 and 833 commits, respectively, as shown in Table 9.

Peril 3 does not apply to our dataset as the projects have a median of one commit per week and because we removed projects with more than two years without commits.

For Peril 4, the authors found that about 10% of the developers used GitHub for storage. This is partially true for our dataset: we found engine repositories that were used mostly to store assets, documentation, and other files.

**Table 5**
Perils of Github repositories adapted from Kalliamvakou et al. [47]. Perils 7, 8, and 9 do not pertain to this work.

| # | Perils | Eng. | Fram. |
|---|---|---|---|
| 1 | A repository is not necessarily a project. | False | False |
| 2 | Most projects have very few commits. | False | False |
| 3 | Most projects are inactive. | False | False |
| 4 | A large portion of repositories are not for software development. | True | False |
| 5 | Two thirds of projects (71.6% of repositories) are personal. | True | False |
| 6 | Only a fraction of projects use pull requests. And of those that use them, their use is very skewed. | True | True |

Peril 5 is present in this study, specifically in game engine projects: although we removed engines with less than 2 contributors, we found many warnings in read-me files stating that an engine was only for "personal use", an "unfinished project", or for "educational purposes" only.

Peril 6 is true for all projects. The number of pull requests for engines is lower than that for frameworks: at least 50% of the engine projects have at least 10 closed pull requests, while frameworks have 100s.

In general, the perils found in any repositories in GitHub do not apply to our dataset. Engines and frameworks seem different to the projects studied by Kalliamvakou et al. [47].

### 6.2. Discussion of RQ1: Literature Perspective

In theory, game engines and frameworks have similar objectives: they are *modular platforms* for *reuse* that provide a *standard* way to develop a product, lowering the barrier of entry for developers by *abstracting* implementation details.

We could classify frameworks in different categories, according to their domains, e.g., Web apps, mobile apps, AI, etc. In a same category and across categories, two frameworks are not the same. They provide their functionalities in different ways. Similarly, game engines also belong to different categories and are different from one another. For example, 3D or 2D and specific for game genres, like *platformer*, *shooter*, *racing*, etc.

Traditional frameworks provide business services while game engines support entertaining games [16]. The process of finding the "fun factor" is exclusive to game development [19, 21] but do not exempt developers from using traditional software-engineering practices [4, 48]. Game engines are tools that help game developers to build games and, therefore, are not directly concerned with non-functional requirements of games, such as "being fun".

Figure 5 shows the relationship between game engines, games, frameworks, and traditional software: a *video game* is a product built on top of a *engine*, like a *Web app* is built on top of a *Web framework*. Engines are a specific kind of

framework used to build games. Everything described is a *software*: Scrumpy[24] is a Web app written with Vue while Dota 2[25] is a game made with Source.

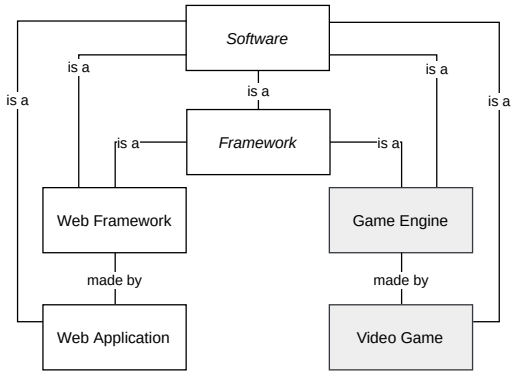



**Figure 5:** Semantic relationship between software, framework, and product.

## 6.3. Discussion of RQ2: Code Perspective

### *6.3.1. RQ2.1: Static Characteristics*

*Differences in Programming Languages.* There is a discrepancy between the languages used in game engines, which belong mostly to the C family, and frameworks, developed mostly with interpreted languages. We explain this difference as follows: engines must work close to the hardware and manage memory for performance. Low-level, compiled languages allow developers to control fully the hardware and memory. Frameworks use languages providing higher-level abstractions, allowing developers to focus on features. Frameworks and engines are tools on which developers build their products, who choose the most effective language for their needs.

This observation highlights the needs for performance in engines, through low-level communication with hardware and memory. With the rise of WebAssembly[26] and the possibility of running compiled code in Web browsers, this observation could change in the near future.

We explain the predominance of C++ for engines by a set of features of this language: abstraction, performance, memory management, platforms support, existing libraries, and community. These features together make C++ a good choice for game developers.

Engines are usually written (or extended) via their main programming language. However, to ease the design, implement, test workflow during production, game developers often add scripting capabilities to their engines. Therefore, when writing a game, game developers may not code directly with low-level languages but use scripts; sometimes with in a specific domain-specific language. For example, Unity, although written in C++, offers scripting capabilities in C#[27] for game developers to build their games. Furthermore, the developer can, possible, finish its game just using this high level language. For any further extension in the game engine they will need to deal with the low level language. On the other hand, frameworks rarely offer scripting capabilities: their products are often written in the same programming languages.

*Similarities in Licenses.* The MIT License is the most used license by both open-source frameworks and open-source engines because it allows reusing and distributing source and–or compiled code in open or proprietary products, with or without changes. Developers can use such these frameworks and engines to create and distribute their software and games without restriction. Also, they can extend or change the code without having to share their intellectual property.

*Similarities in Sizes and Complexities.* Our results show small differences in sizes and complexities between engines and frameworks, yet not enough to consider engines different from frameworks.

The size of a piece of code is a simplistic proxy to its quality. Also, regarding the languages and numbers of files, we expected that larger values for frameworks, given the numbers of configuration files and testing functions. However, we reported that engines are larger in all cases, although by a small margin.

The complexities of the functions was another surprise given the large number of small engines: engines are more complex, although by a small difference.

### *6.3.2. RQ2.2: Historical Characteristics*

Our results showed that 40% of the engines do not have tags, which could mean that they are still under development and no build is available.

However, our dataset contains the most important game engines on GitHub, thus there should be other reasons for the lack of engine releases. During our manual analysis, we found engines with warning messages alerting that they were incomplete, lacking some essential features. Also, we observed that about one third of the engines have only two collaborators. This fact combined with the complexity of engines could explain the difficulty to release a first feature-complete version.

Frameworks are released more often than engines with more commits performed more regularly. There are thus meaningful differences between engines and frameworks, which could be explained by the higher popularity of the frameworks (see next section).

### *6.3.3. RQ2.3: Community Characteristics*

*Differences in Truck Factor.* The truck-factor is 1 for most of the engines (83%). Lavallée and Robillard [49] considered that, in addition to being a threat to a project survival, a low truck-factor causes also delays, as the knowledge is concentrated in one developer only. This concentration further limits adoption by new developers. We believe that low truck-factor values are due to the nature of the engines, i.e., side/hobby projects. In contrast, popular frameworks do not

[24] https://scrumpy.io
[25] http://blog.dota2.com
[26] https://webassembly.org/
[27] https://docs.unity3d.com/Manual/ScriptingSection.html

have such a dependency on single developers.

*Differences in Community Engagement.* We assumed that the numbers of stars for projects in GitHub are a good proxy for their popularity [50]. Surprisingly, engines written in Go and frameworks written in C# are most popular, even though their total numbers are low. JavaScript and C are second and third, respectively. Java is barely present despite its age and general popularity.

### 6.4. Closed-Source vs. Open-Source Game Engines

The great majority of commercial games are written with proprietary, closed-source game engines. Recently, an effort for building a robust and powerful open-source game engine became popular with the Godot project. The Godot game engine was thus used by many indie games[28], some of them of high quality[29].

Open-source tools for game development had a promising start with Doom and its engine. However, the game industry took another route and closed-source engines are commonplace nowadays. Despite the difference in popularity between open-source game engines and traditional, open-source frameworks, we believe that open-source is the right path to follow. It democratizes and allow a soft learning curve for beginners. Also, it will allow the creation of more diverse games.

### 6.5. Threats to Validity

Our results are subject to threat to their internal, construct, and external validity.

*Internal Validity.* We related engines and frameworks with static and historical measures. As previous works, we assumed that these measures represent the characteristics that they measure as perceived by developers. It is possible that other measures would be more relevant and representative for developers' choices and perceptions. We mitigated this threat by exploring different perspectives: *literature*, *code*, and *human*. Also, we divided measures along different aspects (static, historical, and community).

*Construct Validity.* We assumed that we could compare fairly projects in different programming languages, for different domains, and with different purposes, as in various previous studies. We claim that different projects can be compared by considering these projects from three different perspectives: *literature*, *code*, and *human*.

*External Validity.* We studied only open-source projects accessible to other researchers and to provide uniquely identifying information. We also shared on-line[12] all the collected data to mitigate this threat by allowing others to study, reproduce, and complement our results.

*Conclusion Validity.* We did not perform a systematic literature review integrating gray literature available on the Internet. We accept this threat and plan a multivocal literature review in future work. Our study of the literature confirmed that game engines are little studied in academia.

The higher popularity of the frameworks is a concern: the numbers of contributors are larger and could lead to unfair comparisons. We ordered the dataset by the most popular frameworks and engines, so we expected such effect. In the future, we will improve the categorization of our dataset by separating frameworks and engines based on their domains (Web, security, etc., and 2D and 3D games, etc.).

We mined the dataset using the tags of GitHub with which developers classify their projects. For game engines, we used some variations like *game engine*, *game-engine*, or *gameengine*. We may have missed some projects if developers did not use relevant, recognisable tags. For example, the game engine Piston[30], written in Rust by 67 contributors, is not part of our dataset because it was tagged as *"piston, rust, modular-game-engine"*. However, we claim that such engines are rare and their absence does not affect our results based on 282 engines and 282 frameworks.

Regarding our survey, Question 3 is broad and could have mislead developers. Although the requirements are different, developers are still creating the building blocks that will serve to build a product. We mitigated this threat through Questions 1 and 2 and the other two perspectives.

Even after filtering out projects with at less than two contributors, most of the open-source engines are, in fact, personal projects. A few, popular, open-source game engines are used by the majority of the released games. In general, commercial games are built using proprietary/closed-source engines (Unity and Unreal). In future work, we could use other metrics to filter the projects than their numbers of contributors and stars; for example, the numbers and stars of the games released using these engines.

Our conclusions, in particular those stemming from RQ2 and RQ3, depend on the *users* of the analysed projects: a project with a larger user base would certainly receive more bug reports, see more commits and more contributions, etc. Therefore, comparing the top frameworks, which are certainly used in dozens of others projects, and the top game engines, which are mostly "personal" projects, could be unfair to game engines. We must accept this threat in the absence of means to obtain statistics on the user base of GitHub projects. Future work includes possibly using GitHub Insights on a selected set of projects in collaboration with their developers' teams.

Finally, we acknowledge that a great part of game-engine development is closed-source. Therefore, these results might not generalize to game engines overall but should hold true to open-source game engines.

[28] https://itch.io/games/made-with-godot

[29] https://youtu.be/UEDEIksGEjQ

[30] https://github.com/PistonDevelopers/piston

## 7. Conclusion

This paper is a step towards confirming that software-engineering practices apply to game development given their commonalities. It investigated open-source game engines, which form the foundation of video games, and compared them with traditional open-source frameworks. Frameworks are used by developers to ease software development and to focus on their products rather than on implementation details. Similarly, game engines help developers create video games and avoid duplication of code and effort.

We studied open-source game engines along three perspectives: *literature*, *code*, and *human*. Our literature review showed a lack of academic studies about engines, especially their characteristics and architectures. Yet, we showed that, different from what researchers and engine developers think, there are qualitative but no quantitative differences between open-source engines and open-source frameworks. Hence, game engines must be an object of study in software-engineering research in their own right.

We divided the code perspective into three points of view: static code (RQ2.1), history of the projects (RQ2.2), and of their community characteristics (RQ2.3). We studied 282 engines and 282 frameworks from GitHub and contributed with the first corpus of curated open-source engines[12]. We reported no significant difference between engines and frameworks for size and complexity but major differences in terms of popularity and community engagement. The programming languages adoption differed greatly also with engines mostly written in C, C++, and C# and frameworks mostly in JavaScript, PHP, and Python. We observed that engines have shorter histories and fewer releases than frameworks.

Finally, our survey results showed that engine developers have also experience in developing traditional software and that they believe that game engines are different from frameworks. The developers' objectives for developing engines are (a) better control the environment and source code, (b) learn, and (c) develop specific games.

> We conclude that open-source game engines share similarities with open-source frameworks, mostly regarding their concepts, code characteristics, and contribution effort. Yet, while engines projects are mainly personal, the communities around framework projects are larger, with longer lifespans, more releases, better truck-factor, and more popularity.
>
> Therefore, engine developers should adopt the similar software code-quality toolkit when dealing with code. Finally, the low truck-factor and smaller user-base suggests that more care should be given to the documentation of the open-source engines.

We will also consider contacting a subset of the top projects and work with their developers' teams so that they install GitHub Insights, which would provide further information on the projects, unavailable at the moment when analysing GitHub data as "outsiders" to the projects.

Some engines appears suitable for a deeper investigation of their core architectures. The outliers are good candidates to find anti-patterns related to engines and frameworks. While Gregory [7] presented a complex description of the architecture of an engine, it would be interesting to see how a real, successful engine architecture is similar to the one proposed by the author. Also, we did not discuss in details the most popular, closed-source engines: Unity and Unreal. We could also study the differences between engines and frameworks regarding their workflow to reveal new differences between both types of software. Finally, further investigate engines and frameworks communities (developers' turnover and how teams are geo-dispersed) as well as why and how these projects choose their languages.

## Acknowledgement

The authors thank all the anonymous developers for their time. The authors were partly supported by the NSERC Discovery Grant and Canada Research Chairs programs.

## References


[1] Jason Schreier. How bioware's anthem went wrong. https://kotaku.com/how-biowares-anthem-went-wrong-1833731964, 2019. Accessed: 2019-04-03.

[2] Entertainment Software Association - ESA. Esa's essential facts about the computer and video game industry report, 2019. [Online; accessed 7-February-2019].

[3] Newzoo. 2019 global games market report. https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2019-light-version/, 2019. [Online; accessed 1-October-2019].

[4] Fábio Petrillo, Marcelo Pimenta, Francisco Trindade, and Carlos Dietrich. What went wrong? a survey of problems in game development. *Computers in Entertainment*, 7(1):1, February 2009. doi: 10.1145/1486508.1486521.

[5] Dayi Lin, Cor Paul Bezemer, and Ahmed E. Hassan. Studying the urgent updates of popular games on the Steam platform. *Empirical Software Engineering*, 22(4):2095–2126, 2017. ISSN 1573-7616. doi: 10.1007/s10664-016-9480-2.

[6] Henrik Edholm, Mikaela Lidstrom, Jan-Philipp Steghofer, and Hakan Burden. 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)*, pages 43–52. IEEE, May 2017. ISBN 9781538627174. doi: 10.1109/icse-seip.2017.18.

[7] Jason Gregory. *Game Engine Architecture, Second Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014. ISBN 1466560010, 9781466560017.

[8] A. Thorn. *Game Engine Design and Implementation*. Foundations of game development. Jones & Bartlett Learning, 2011. ISBN 9780763784515.

[9] Jason Hughes. What to look for when evaluating middleware for integration. In Eric Lengyel, editor, *Game Engine Gems 1*, pages 3–10. Jones and Bartlett, 2010.

[10] A. Sherrod. *Ultimate 3D Game Engine Design & Architecture*. Charles River Media game development series. Charles River Media, 2007. ISBN 9781584504733.

[11] Jason Schreier. The controversy over bethesda's 'game engine' is misguided. https://kotaku.com/the-controversy-over-bethesdas-game-engine-is-misguided-1830435351, 2018. Accessed: 2019-02-06.

[12] Farouk Messaoudi, Gwendal Simon, and Adlen Ksentini. Dissecting games engines: The case of Unity3D. *Annual Workshop on Network and Systems Support for Games*, 2016-January:1–6, 2016. ISSN 2156-8146. doi: 10.1109/NetGames.2015.7382990.

[13] J. Schreier. *Blood, Sweat, and Pixels: The Triumphant, Turbulent Stories Behind How Video Games Are Made*. HarperCollins, 2017. ISBN 9780062651242.

[14] D. Kushner. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House Publishing Group, 2003. ISBN 9781588362896.

[15] Fabio Petrillo and Marcelo Pimenta. Is agility out there? In *Proceedings of the 28th ACM International Conference on Design of Communication - SIGDOC 10*, pages 9–15. ACM Press, 2010. doi: 10.1145/1878450.1878453.

[16] 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)*, pages 15–21. IEEE, May 2017. doi: 10.1109/wetsom.2017.3.

[17] Sami Hyrynsalmi, Eriks Klotins, Michael Unterkalmsteiner, Tony Gorschek, Nirnaya Tripathi, Leandro Bento Pompermaier, and Rafael Prikladnicki. What is a Minimum Viable (Video) Game? In *17th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2018*, volume 11195, pages 217–231. Springer-Verlag GmbH, October 2018. ISBN 3030021300. doi: 10.1007/978-3-030-02131-3.

[18] Emerson Murphy-Hill, Thomas Zimmermann, and Nachiappan Nagappan. Cowboys, Ankle Sprains, and Keepers of Quality: How is Video Game Development Different from Software Development? *Proceedings of the 36th International Conference on Software Engineering*, pages 1–11, 2014. doi: 10.1145/2568225.2568226.

[19] Chris Lewis and Jim Whitehead. The whats and the whys of games and software engineering. In *Proceeding of the 1st international workshop on Games and software engineering - GAS 11*, pages 1–4. ACM Press, 2011. ISBN 9781450305785. doi: 10.1145/1984674.1984676.

[20] Christopher M. Kanode and Hisham M. Haddad. Software engineering challenges in game development. In *2009 Sixth International Conference on Information Technology: New Generations*, pages 260–265. IEEE, 2009. ISBN 9780769535968. doi: 10.1109/itng.2009.74.

[21] David Callele, Philip Dueck, Krzysztof Wnuk, and Peitsa Hynninen. Experience requirements in video games: Definition and testability. *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, pages 324–333, 2015. doi: 10.1109/RE.2015.7320449.

[22] Rido Ramadan and Bayu Hendradjaya. Development of game testing method for measuring game quality. In *2014 International Conference on Data and Software Engineering (ICODSE)*, pages 1–6. IEEE, November 2014. ISBN 9781479979967. doi: 10.1109/icodse.2014.7062694.

[23] Maxim Mozgovoy and Evgeny Pyshkin. A comprehensive approach to quality assurance in a mobile game project. In *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia on ZZZ - CEE-SECR 18*, pages 1–8. ACM Press, 2018. ISBN 9781450361767. doi: 10.1145/3290621.3290835.

[24] Marcus Toftedahl and Henrik Engström. A Taxonomy of Game Engines and the Tools that Drive the Industry. In *Proceedings of the 2019 DiGRA International Conference: Game, Play and the Emerging Ludo-Mix*, pages –, 2019.

[25] G. Avelino, L. Passos, A. Hora, and M. T. Valente. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, May 2016. doi: 10.1109/ICPC.2016.7503718.

[26] Barbara A. Kitchenham. Systematic review in software engineering: Where we are and where we should be going. In *Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies*, EAST '12, pages 1–2. Association for Computing Machinery, 2012. ISBN 9781450315098. doi: 10.1145/2372233.2372235.

[27] Wolfgang Pree. Meta patterns—a means for capturing the essentials of reusable object-oriented design. In *European Conference on Object-Oriented Programming*, pages 150–162. Springer, 1994.

[28] Craig Larman. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012.

[29] Richard N. Taylor. Only the architecture you need. In Volker Gruhn and Rüdiger Striemer, editors, *The Essence of Software Engineering*, pages 77–89. Springer International Publishing, Cham, 2018. ISBN 978-3-319-73897-0. doi: 10.1007/978-3-319-73897-0_5.

[30] Henry Lowood and Raiford Guins. *Debugging Game History: A Critical Lexicon*. The MIT Press, 2016. ISBN 0262034190, 9780262034197.

[31] Henry Lowood. Game Engines and Game History. *Kinephanos: History of Games International Conference Proceedings*, pages 179–98, January 2014.

[32] Michael Lewis and Jeffrey Jacobson. Introduction. *Communications of the ACM*, 45(1):27–31, January 2002. ISSN 0001-0782. doi: 10.1145/502269.502288.

[33] Brent Cowan and Bill Kapralos. A survey of frameworks and game engines for serious game development. In *2014 IEEE 14th International Conference on Advanced Learning Technologies*, pages 662–664. IEEE, July 2014. ISBN 9781479940387. doi: 10.1109/icalt.2014.194.

[34] Brent Cowan and Bill Kapralos. An overview of serious game engines and frameworks. In Anthony Lewis Brooks, Sheryl Brahnam, Bill Kapralos, and Lakhmi C. Jain, editors, *Recent Advances in Technologies for Inclusive Well-Being*, pages 15–38. Springer Interna-

tional Publishing, 2017. ISBN 978-3-319-49879-9. doi: 10.1007/978-3-319-49879-9_2.

[35] Mario Popolin Neto and Jose Remo Ferreira Brega. A survey of solutions for game engines in the development of immersive applications for multi-projection systems as base for a generic solution design. In *2015 XVII Symposium on Virtual and Augmented Reality*, pages 61–70. IEEE, May 2015. ISBN 9781467372046. doi: 10.1109/svr.2015.16.

[36] Alf Inge Wang and Njål Nordmark. Software architectures and the creative processes in game development. In Konstantinos Chorianopoulos, Monica Divitini, Jannicke Baalsrud Hauge, Letizia Jaccheri, and Rainer Malaka, editors, *Entertainment Computing - ICEC 2015*, pages 272–285, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24589-8.

[37] Eike Falk Anderson, Steffen Engel, Peter Comninos, and Leigh McLoughlin. The case for research in game engine architecture. In *Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play 08*, page 228. ACM Press, 2008. ISBN 9781605582184. doi: 10.1145/1496984.1497031.

[38] Antonio Lima, Luca Rossi, and Mirco Musolesi. Coding together at scale: GitHub as a collaborative social network. *Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014*, pages 295–304, 2014.

[39] Laurie Williams and Robert Kessler. *Pair programming illuminated*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[40] Cem Kaner, Senior Member, and Walter P. Bond. Software engineering metrics: What do they measure and how do we know? In *In METRICS 2004. IEEE CS*. Press, 2004.

[41] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 131–142. Association for Computing Machinery, 2008. ISBN 9781605580500. doi: 10.1145/1390630.1390648.

[42] T. DeMarco. *Controlling Software Projects: Management, Measurement, and Estimates*. Prentice Hall PTR, USA, 1986. ISBN 0131717111.

[43] T. J. McCabe. A Complexity Measure. *IEEE Transaction Software Engineering*, SE-2(4):308–320, December 1976. ISSN 1939-3520. doi: 10.1109/TSE.1976.233837.

[44] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6):772–787, 2011.

[45] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, L. Gren, Carlo A. Furia, and Z. Huang. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *Journal of Systems and Software*, 156:246–267, 2019. ISSN 0164-1212. doi: 10.1016/j.jss.2019.07.002.

[46] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. ISBN 3642290434.

[47] Eirini Kalliamvakou, Leif Singer, Georgios Gousios, Daniel M. German, Kelly Blincoe, and Daniela Damian. The promises and perils of mining GitHub. *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings*, pages 92–101, 2014. doi: 10.1145/2597073.2597074.

[48] Jussi Kasurinen. Games as software. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 - CompSysTech 16*, pages 33–40. ACM Press, 2016. ISBN 9781450341820. doi: 10.1145/2983468.2983501.

[49] Mathieu Lavallée and Pierre N. Robillard. Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 677–687. IEEE Press, 2015. ISBN 978-1-4799-1934-5. URL http://dl.acm.org/citation.cfm?id=2818754.2818837.

[50] Hudson Borges and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018. ISSN 0164-1212. doi: 10.1016/j.jss.2018.09.016.

[51] Christopher Vendome, Gabriele Bavota, Massimiliano Di Penta, Mario Linares-Vásquez, Daniel German, and Denys Poshyvanyk. *License usage and changes: a large-scale study on gitHub*, volume 22. Springer, 2017. ISBN 1066401694384. doi: 10.1007/s10664-016-9438-4.

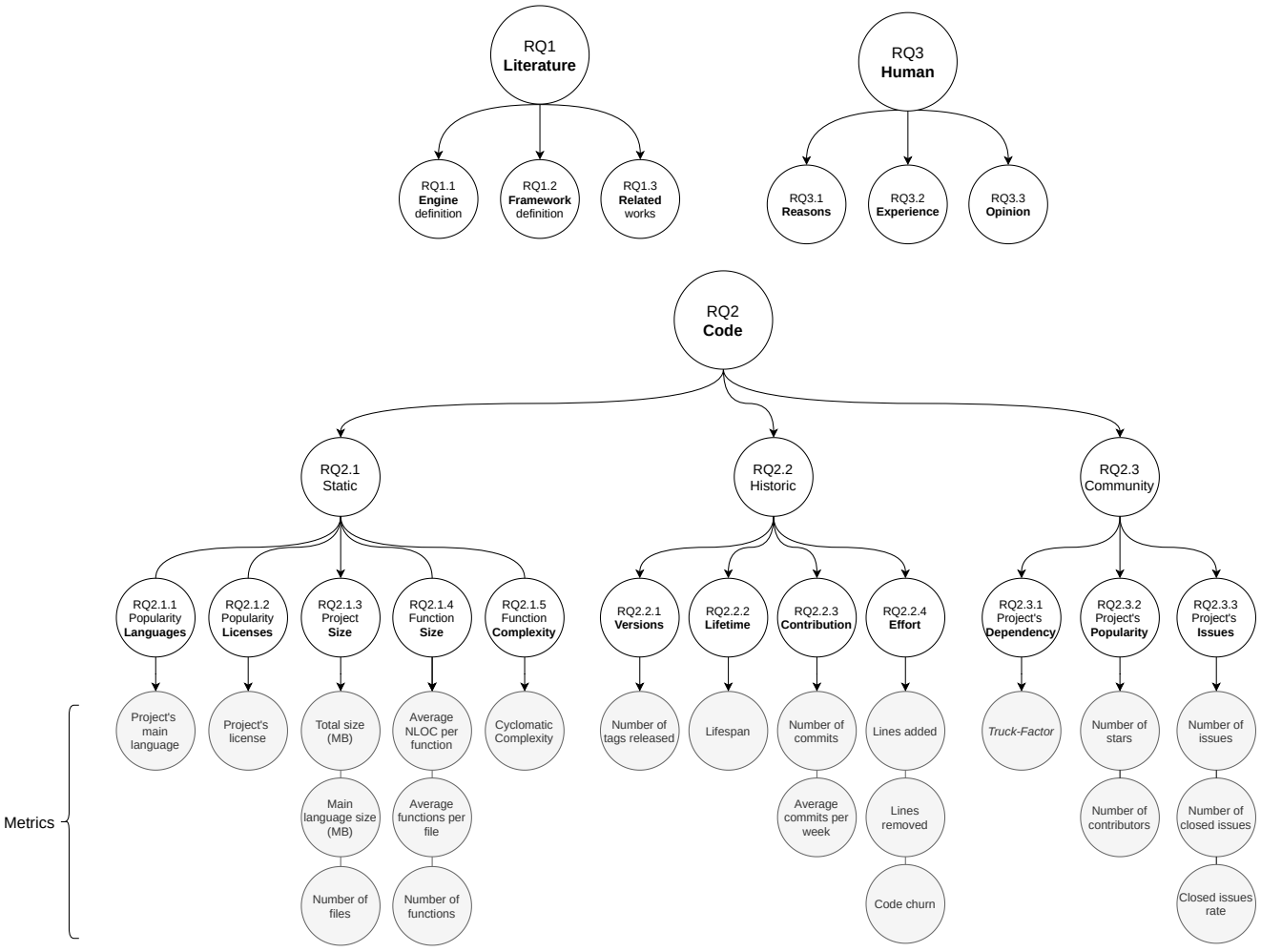



**Figure 6:** The three perspectives of the study, the research questions and metrics.

# Appendix

## A. Detailed Results for RQ2.1: Static Characteristics

### RQ2.1.1: What is the popularity of the *languages* in the projects?

Table 7 shows the popularity of the programming languages in both framework and game engines, ordered by the numbers of projects. The most used languages in game engines belong to the C family: C, C++, and C#. Together, they represent about 64% of the code. For frameworks, JavaScript, PHP, and Python are the most used languages with 51% of the code. C++ and JavaScript are the most used language for games and frameworks.

The distributions of the languages differ for each group. Game engines are mainly built in C++. For frameworks, the differences between the top three languages are smaller. Because we sorted projects by popularity and most top frameworks focus on Web development, interpreted languages are used in the majority of their code.

Figure 7 shows the ranking of the top six most used languages in engines and frameworks and compare it to other three global sources of programming languages usage in 2019: GitHut 2.0[31] (Third Quarter), which uses the GitHub API to query the most used languages in the public repositories, Tiobe index[32] (November), which uses a set of metrics together with results from search engines, and PYPL[33] (PopularitY of Programming Language, December), which is a ranking created by analysing how often language tutorials are searched on Google.

C++ is the most popular language for engines but is only the 10$^{th}$ most popular in frameworks, 5$^{th}$ in GitHut, 4$^{th}$ in Tiobe, and 6$^{th}$ in PYPL. C is used in engines and is in 2$^{nd}$ position in Tiobe but not so popular according to the other sources. JavaScript is embraced by the open-source community and received lots of attention in searches but Tiobe puts it in the 7$^{th}$ place. The popularity of the programming languages in engines is more aligned to the Tiobe index than to the other sources. In contrast, popular languages in frameworks are more aligned to GitHut and PYPL rankings. Game engines are more aligned with the commercial market and

[31] https://madnight.github.io/githut/
[32] https://www.tiobe.com/tiobe-index/
[33] https://pypl.github.io/PYPL.html

**Table 6**
Descriptive Statistics, RQ2.1: Static Characteristics. Normality <0.01 means the data is not normally distributed.

| RQs | Variable | Type | Mean | Std.Dev. | Median | Min | Max | Normality |
|---|---|---|---|---|---|---|---|---|
| | main_language_size | engine | 5.78 | 14.95 | 1.09 | 0.00 | 102.10 | <0.01 |
| | main_language_size | framework | 3.82 | 17.74 | 0.55 | 0.00 | 276.76 | <0.01 |
| RQ2.3 | total_size | engine | 7.66 | 20.31 | 1.22 | 0.00 | 155.32 | <0.01 |
| | total_size | framework | 4.82 | 26.27 | 0.60 | 0.00 | 423.37 | <0.01 |
| | n_file | engine | 685.60 | 1853.12 | 171.00 | 1.00 | 23379.00 | <0.01 |
| | n_file | framework | 456.93 | 1053.45 | 97.50 | 1.00 | 8062.00 | <0.01 |
| | n_func | engine | 10130.74 | 21627.98 | 2394.00 | 1.00 | 163779.00 | <0.01 |
| | n_func | framework | 5924.17 | 14469.19 | 960.50 | 1.00 | 145288.00 | <0.01 |
| RQ2.4 | nloc_mean | engine | 12.94 | 15.17 | 11.07 | 1.32 | 247.25 | <0.01 |
| | nloc_mean | framework | 12.71 | 33.89 | 8.79 | 1.00 | 539.79 | <0.01 |
| | func_per_file_mean | engine | 20.60 | 40.32 | 12.34 | 1.00 | 370.14 | <0.01 |
| | func_per_file_mean | framework | 23.03 | 82.62 | 8.57 | 1.00 | 1070.13 | <0.01 |
| RQ2.5 | cc_mean | engine | 3.09 | 2.35 | 2.77 | 1.00 | 36.19 | <0.01 |
| | cc_mean | framework | 2.68 | 3.75 | 2.14 | 1.00 | 60.22 | <0.01 |

**Table 7**
Popularity of programming languages among engines and frameworks.

| | Engine | | Framework | | Total | |
|---|---|---|---|---|---|---|
| | N | % | N | % | N | % |
| C++ | 107 | 37.94% | 10 | 3.55% | 117 | 20.74% |
| JavaScript | 28 | 9.93% | 71 | 25.18% | 99 | 17.55% |
| Python | 14 | 4.96% | 45 | 15.96% | 59 | 10.46% |
| C | 41 | 14.54% | 11 | 3.90% | 52 | 9.22% |
| PHP | 3 | 1.06% | 46 | 16.31% | 49 | 8.69% |
| C# | 33 | 11.70% | 15 | 5.32% | 48 | 8.51% |
| Java | 27 | 9.57% | 19 | 6.74% | 46 | 8.16% |
| Go | 14 | 4.96% | 21 | 7.45% | 35 | 6.21% |
| TypeScript | 7 | 2.48% | 18 | 6.38% | 25 | 4.43% |
| Swift | 2 | 0.71% | 13 | 4.61% | 15 | 2.66% |
| Scala | 1 | 0.35% | 5 | 1.77% | 6 | 1.06% |
| Objective-C | 1 | 0.35% | 4 | 1.42% | 5 | 0.89% |
| Lua | 4 | 1.42% | 0 | 0.00% | 4 | 0.71% |
| Ruby | 0 | 0.00% | 4 | 1.42% | 4 | 0.71% |

**Table 8**
Most Used Licenses.

| | Engine | | Framework | | Total | |
|---|---|---|---|---|---|---|
| | N | % | N | % | N | % |
| MIT License | 116 | 41% | 142 | 50% | 258 | 46% |
| Other | 55 | 20% | 48 | 17% | 103 | 18% |
| *No licence specified* | 30 | 11% | 22 | 8% | 52 | 9% |
| Apache License 2.0 | 17 | 6% | 35 | 12% | 52 | 9% |
| GNU GPL v3.0 | 28 | 10% | 10 | 4% | 38 | 7% |
| GNU LGPL v3.0 | 8 | 3% | 5 | 2% | 13 | 2% |
| BSD 3 | 2 | 1% | 7 | 2% | 9 | 2% |
| GNU GPL v2.0 | 7 | 2% | 2 | 1% | 9 | 2% |
| zlib License | 6 | 2% | 0 | 0% | 6 | 1% |
| GNU AGPL v3.0 | 2 | 1% | 3 | 1% | 5 | 1% |
| BSD 2 | 3 | 1% | 1 | 0% | 4 | 1% |
| GNU LGPL v2.1 | 1 | 0% | 3 | 1% | 4 | 1% |
| Mozilla PL 2.0 | 2 | 1% | 1 | 0% | 3 | 1% |
| The Unlicense | 2 | 1% | 1 | 0% | 3 | 1% |
| Artistic License 2.0 | 1 | 0% | 0 | 0% | 1 | 0% |
| Boost SL 1.0 | 0 | 0% | 1 | 0% | 1 | 0% |
| CC Attribution 4.0 | 1 | 0% | 0 | 0% | 1 | 0% |
| Eclipse PL 1.0 | 0 | 0% | 1 | 0% | 1 | 0% |
| Microsoft PL | 1 | 0% | 0 | 0% | 1 | 0% |

less with open-source projects.

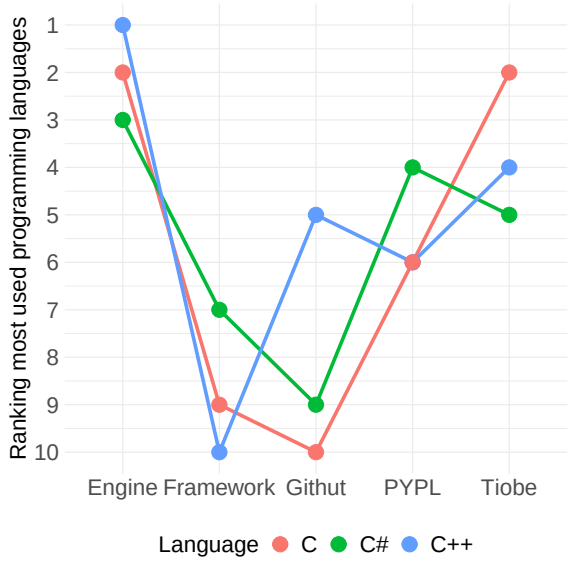
### RQ2.1.2: What is the popularity of the *licenses* in the projects?

Table 8 shows the top 10 most used licenses. Differently from the languages, the distribution of licenses is similar between engines and frameworks. The MIT License is most used for both types of projects with 46%. "Other" licenses (not reported by GitHub) are the second most popular with 18%. 9% of the projects do not have an explicit license. The remaining ones form 27%.

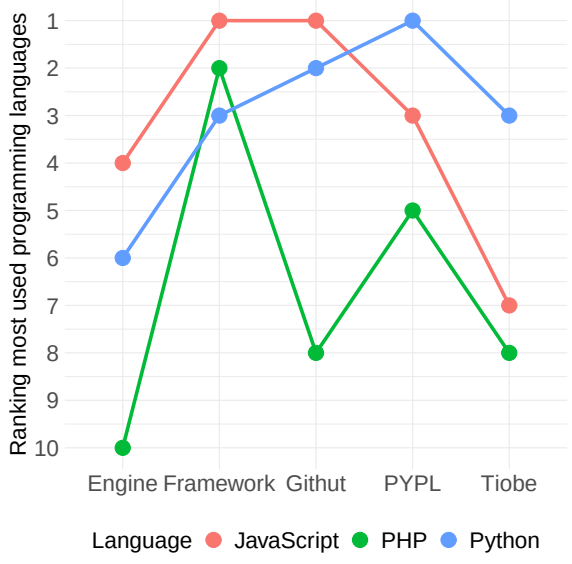
According to GitHut 2.0, the licenses in GitHub project are ranked as follows: MIT 54%, Apache 16%, GPL 2 13%, GPL 3 10%, and BSD 3 5%, which is similar to the rankings in engines and frameworks. Projects with "Other" licenses are a big part of this data, which, according to Vendome et al. [51], are prone to migrate towards Apache or GPL licenses. Finally, the MIT license is popular thanks to its permissive model, which fits well with most open-source projects.

The licenses might only apply to the engines or frameworks and *not* to the games or software. For example, games created with Godot have their creators as sole copyright owners but must include its license:

> *"Godot Engine's license terms and copyright do not apply to the content you create with it; you are free to license your games how you see*




(a) Most used languages in Game Engines.



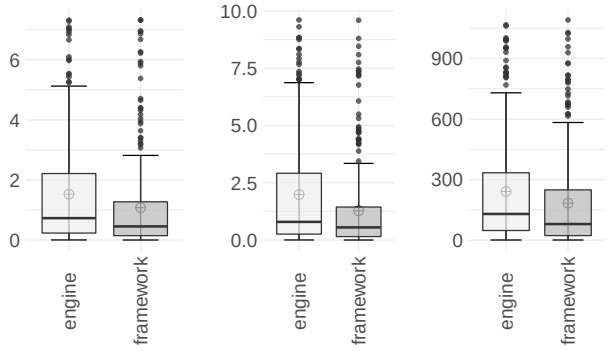

(b) Most used languages in Frameworks.

**Figure 7:** Ranking of the languages used in engines and frameworks compared to their global uses.

> *best fit, and will be their sole copyright owner(s). Note however that the Godot Engine binary that you would distribute with your game is a copy of the 'Software' as defined in the license, and you are therefore required to include the copyright notice and license statement somewhere in your documentation."*
>
> – `https://godotengine.org/license`

### RQ2.1.3: What are the *project sizes* of engines and frameworks?

We considered *main_language_size*, *total_size*, and *n_files*. They show larger values for engines when compared to frameworks. Considering the medians, engines have around 50% higher median values regarding size of the main language (1.09MB), total size of the project (1.22MB), and number of files (171 files). The boxplots in Figure 8 help to identify the differences among variables: game engines are larger than frameworks, on average.

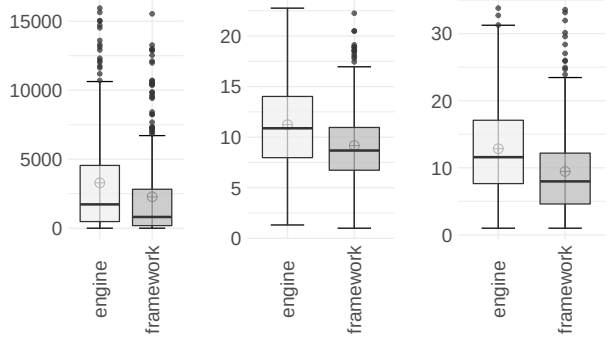



(a) Language size (b) Total size (c) Number of files

**Figure 8:** Boxplots – RQ2.1.3: What are the *project sizes* of engines and frameworks?
(a) `main_language_size`
(b) `total_size`
(c) `n_files`

### RQ2.1.4: What are the *function sizes* of engines and frameworks?

We considered *n_func*, *nloc_mean*, and *func_per_file_mean*. The boxplots in Figure 9 show that engines have larger values when compared to frameworks. Considering medians, engines have around 30% more functions per file and 20% more functions and lines of code per function.




(a) Functions (b) Code lines (c) Functions per file

**Figure 9:** Boxplots – RQ2.1.4: What are the *function sizes* of engines and frameworks?
(a) `n_funcs`
(b) `nloc_mean`
(c) `func_per_file_mean`

### RQ2.1.5: What are the *function complexities* of engines and frameworks?

We considered *cc_mean* to assess functions complexities. The median for engines is about 23% greater than the frameworks, which correspond to a complexity of less than 1. Figure 10 illustrates this difference. We also identified 16 projects (10 engines and 6 frameworks) with median complexities greater than 5.

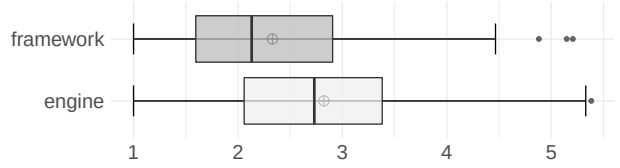



**Figure 10:** Boxplot of the Cyclomatic Complexity (CC).

Factoring out low-level languages, e.g., C++, the complexity of the functions are similar between engines and frameworks. The number of files and average functions per file are similar. The total number of functions, however, is different: engines have twice as many functions (median values).

Programming languages vary greatly, as the game engines are written mostly in compiled languages, while frameworks in interpreted ones. Both types of projects prefer the MIT license, very suitable to open-source projects. Although game engines are, on average, bigger and more complex than frameworks, this difference is small.

**Table 9**
Descriptive Statistics: RQ2.2: Historical Characteristics. Normality <0.01 means the data is not normally distributed.

| RQs | Variable | Type | Mean | Std.Dev. | Median | Min | Max | Normality |
|---|---|---|---|---|---|---|---|---|
| RQ3.1 | tags_releases_count | engine | 15.82 | 52.20 | 1.00 | 0.00 | 657.00 | <0.01 |
| | tags_releases_count | framework | 82.24 | 216.37 | 32.00 | 0.00 | 2,678.00 | <0.01 |
| RQ3.2 | lifespan (weeks) | engine | 155.70 | 113.39 | 135.79 | 0.00 | 530.43 | <0.01 |
| | lifespan (weeks) | framework | 215.30 | 129.79 | 182.14 | 5.71 | 590.71 | <0.01 |
| RQ3.3 | commits_count | engine | 2,029.93 | 4,553.92 | 616.00 | 7.00 | 37,026.00 | <0.01 |
| | commits_count | framework | 3,463.88 | 8,581.04 | 833.50 | 20.00 | 87,774.00 | <0.01 |
| | commits_per_time | engine | 3.44 | 7.71 | 1.04 | 0.01 | 62.68 | <0.01 |
| | commits_per_time | framework | 5.86 | 14.53 | 1.41 | 0.03 | 148.59 | <0.01 |
| RQ3.4 | lines_added | engine | 2,403.59 | 9,179.15 | 424.20 | 7.00 | 94,597.77 | <0.01 |
| | lines_added | framework | 776.62 | 2,750.07 | 169.46 | 5.80 | 26,099.63 | <0.01 |
| | lines_removed | engine | 644.34 | 1,343.49 | 175.44 | 0.00 | 11,957.13 | <0.01 |
| | lines_removed | framework | 434.67 | 1,421.13 | 104.84 | 2.33 | 15,450.48 | <0.01 |
| | code_churn | engine | 3,074.94 | 9,512.87 | 423.86 | 7.00 | 95,426.69 | <0.01 |
| | code_churn | framework | 1,211.28 | 4,015.20 | 163.79 | 10.67 | 41,550.10 | <0.01 |

## B. Detailed Results for RQ2.2: Historical Characteristics

### RQ2.2.1: How many versions were *released* for each project?

Around 40% of the engines (112 projects) do not have any tag. Only 8% of the frameworks (23 projects) are lacking them. Most engines have between 0 and 11 tags while frameworks have between 9 to 88. Frameworks release new versions more often. Figure 11 shows the boxplots for the numbers of tags. The dots represent the outliers as the majority of the engines have zero or few tags.

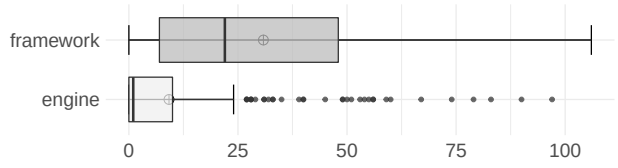



**Figure 11:** Boxplot of number of tags – RQ2.2.1: How many versions were *released* for each project?

### RQ2.2.2: What is the *lifetime* of the projects?

Figure 12 shows the distributions of engines and frameworks lifetimes in weeks: both have similar shapes, with more projects in the last years. Considering median values, engines and frameworks are 2.6 and 3.5 years-old, respectively. Open-source engines are more recent when compared to open-source frameworks.

### RQ2.2.3: How frequently do projects receive new *contributions*?

Figure 13a presents the distribution for *commits_count*. The frequency and number of commits is larger for frameworks in total. Most engines have more than 616 commits, while frameworks have more than 833, in the median.

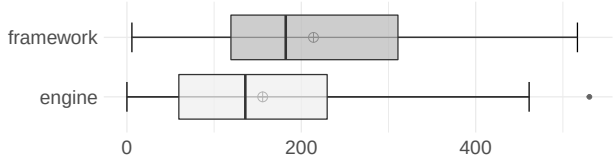
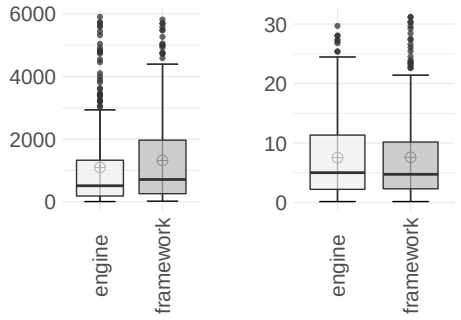



**Figure 12:** Boxplot of lifespan in weeks – RQ2.2.2: What is the *lifetime* of the projects?

Figure 13b depicts the distribution for *commits_per_time*, which represents the number of commits (*commits_count*) averaged by the projects' lifetime (*lifespan*). Interestingly, engines are more active than frameworks when considering the number of commits overtime. In fact, 47% of the engines have at least one commit per week, while this activity is achieved by 40% of frameworks. This behavior—together with the results presented in RQ2.2—may indicate that engines are less mature than frameworks.

### RQ2.2.4: Are commits made on game engines more *effort-prone*?

To answer this research question we investigated three different metrics: *lines_added*, *lines_removed*, and *code_churn*. To keep our comparison at project level, we averaged the values of these metrics based on the number of commits each project has. Overall, engines added and removed more lines than frameworks. In median basis, 424.20 lines were added by engines, against 169.46 lines for frameworks. When it comes to lines removed, engines deleted 175.44 of them, while frameworks removed 104.84; also median values. This results in a code churn that is 2.6x times higher for engines when compared to frameworks (423.86 vs 163.79), which




(a) Commits (b) Commits per week

**Figure 13:** Boxplots – RQ2.2.3: How frequently do projects receive new *contributions*?

means that engines are more likely to have major changes than frameworks.

**Table 10**
Descriptive Statistics, RQ2.3: Community Characteristics. Normality <0.01 means the data is not normally distributed.

| RQs | Variable | Type | Mean | Std.Dev. | Median | Min | Max | Normality |
|---|---|---|---|---|---|---|---|---|
| RQ4.1 | truck_factor | engine | 1.32 | 0.91 | 1 | 1 | 8 | <0.01 |
| | truck_factor | framework | 1.58 | 1.88 | 1 | 1 | 25 | <0.01 |
| RQ4.2 | stargazers_count | engine | 659.45 | 2,140.45 | 44.5 | 2 | 23775 | <0.01 |
| | stargazers_count | framework | 4,017.36 | 11,671.63 | 556.5 | 111 | 145516 | <0.01 |
| | contributors_count | engine | 18.95 | 52.66 | 3 | 2 | 435 | <0.01 |
| | contributors_count | framework | 57.09 | 94.34 | 15 | 2 | 403 | <0.01 |
| RQ4.3 | issues_count | engine | 494.19 | 2,453.87 | 44 | 0 | 30,317 | <0.01 |
| | issues_count | framework | 1,534.70 | 4,075.97 | 254 | 3 | 36,757 | <0.01 |
| | closed_issues_count | engine | 428.5 | 2,122.72 | 34.5 | 0 | 24,861 | <0.01 |
| | closed_issues_count | framework | 1,449.18 | 3,496.88 | 215.5 | 0 | 35,659 | <0.01 |
| | closed_issues_rate | engine | 80% | 20% | 85% | 0% | 100% | <0.01 |
| | closed_issues_rate | framework | 87% | 14% | 91% | 0% | 100% | <0.01 |

## C. Detailed Results for RQ2.3: Community Characteristics

### RQ2.3.1: How many developers *contribute* in the project?

Table 11 shows the truck-factor values and numbers of contributors per project. The distribution of the truck-factor between engines and frameworks are similar with the majority of the projects having a value equal to one (82% for engines and 73% for frameworks). The engine with the highest truck-factor is PGZero with value of 8. Three frameworks have truck-factor values higher than 8: Django (9), Rails (13), and FrameworkBenchmarks (25). As a comparison, Linux[34] has a truck-factor of 57 and Git of 12 [25].

**Table 11**
Truck-factor values and medians of contributors.

| | Frameworks | | Engines | |
|---|---|---|---|---|
| Truck-factor | N | Contributors | N | Contributors |
| 1 | 208 | 11 | 231 | 3 |
| 2 | 46 | 46 | 35 | 13 |
| 3 | 10 | 75.5 | 7 | 47 |
| 4 | 11 | 75 | 4 | 50 |
| 5 | 1 | 355 | 1 | 216 |
| 6 | 2 | 299 | 1 | 312 |
| 7 | – | – | 2 | 294 |
| 8 | 1 | 69 | 1 | 32 |
| 9 | 1 | 403 | – | – |
| 13 | 1 | 377 | – | – |
| 25 | 1 | 374 | – | – |

The median of contributors follows an direct relation: the higher the truck-factor, the higher the number of contributors. The exceptions are the engine PGZero (Python) with 32 contributors and the framework Sofa (C++) with 69 contributors, both with truck-factor 8.

[34] https://github.com/torvalds/linux

### RQ2.3.2: How *popular* are the projects considering their main languages?

Figure 14 shows the popularity of the projects considering the top 10 most used languages (Table 7) ordered by median numbers of stars. Engines written in Go have the highest popularity although they are only 14. JavaScript is the second most popular language followed by the C family. Although C++ makes up the majority of the engines, it is only the fifth most popular. C# is the most popular language for frameworks, but with only 15 projects. JavaScript and C are second and third, respectively.

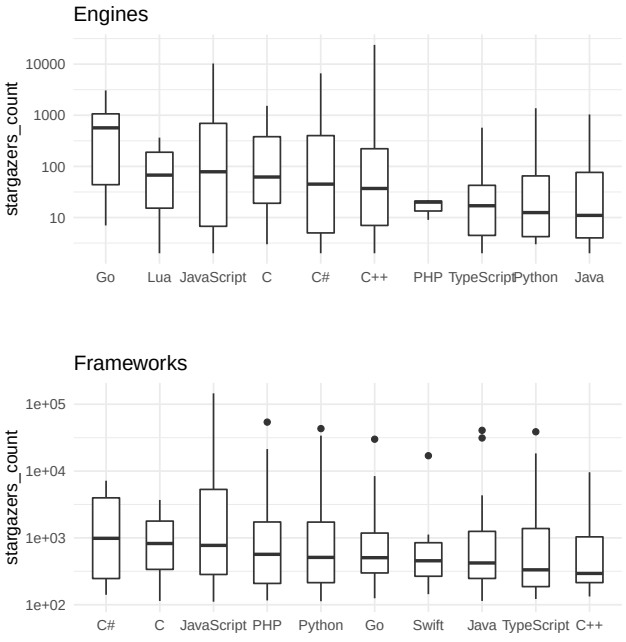



**Figure 14:** Popularity of the Top 10 Most Used Languages for Engines and Frameworks Ordered by Medians.

### RQ2.3.3: How many *issues* are reported in each project?

As observed in Table 10, issues activity on frameworks are higher when compared to engines. For instance, 50% of the frameworks have at least 254 issues reported (i.e., median). This number drops to 44 for engines. The number of closed issues present a similar difference: half of the projects have at least 215 and 34 issues closed for frameworks and engines, respectively. When it comes to the rate of closed issues, both kinds of systems present similar results, though. In median, engines have closed 85% of the issues reported so far, while frameworks have closed 91% of them.