A Case Study on the Availability of Open-Source Components for Game Development

Maria-Eleni Paschali[1](✉), Apostolos Ampatzoglou[2], Stamatia Bibi[3], Alexander Chatzigeorgiou[4], and Ioannis Stamelos[1]

[1] Department of Computer Science, Aristotle University of Thessaloniki, Thessaloniki, Greece
{mpaschali, stamelos}@csd.auth.gr

[2] Institute of Mathematics and Computer Science, University of Groningen, Groningen, The Netherlands
a.ampatzoglou@rug.nl

[3] Department of Informatics and Telecommunications, University of Western Macedonia, Kozani, Greece
sbibi@uowm.gr

[4] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
achat@uom.gr

Abstract. Nowadays the amount of source code that is freely available inside open-source software repositories offers great reuse opportunities to software developers. Therefore, it is expected that the implementation of several requirements can be facilitated by reusing open source software components. In this paper, we focus on the reuse opportunities that can be offered in one specific application domain, i.e., game development. In particular, we performed an embedded multiple case study on approximately 110 open-source games, exploiting a large-scale repository of OSS components, and investigated: (a) which game genres can benefit from open source reuse, and (b) what types of requirements can the available open-source components map to. The results of the case study suggest that: (a) game genres with complex game logic, e.g., First Person Shooter, Strategy, Role-Playing, and Sport games offer the most reuse opportunities, and (b) the most common requirement types that can be developed by reusing OSS components are related to scenarios and characters.

1 Introduction

The last two decades video games have become one of the most important forms of entertainment in modern societies, with respect to their social and economic impact. Specifically, in recent years, and especially among the youth, playing games has outperformed many other types of entertainment, like listening to music or watching movies. Additionally, it is reported that the worldwide revenue of the game industry increased from nearly \$11 billion in 2003 to \$50 billion in 2007 [13] and is still rising until now. One of the most important business requirements of successful game series,

DOI: 10.1007/978-3-319-35122-3_11

which is a prerequisite for surviving demanding competition, is the need for continuous release of newer game versions or patches. Therefore, game development is an intense process, which requires techniques that will shorten the product time to market and simultaneously minimize the effort spent for debugging and testing activities [3, 30].

Reuse is a software engineering technique that offers such benefits, since it increases development productivity [8, 32] and product quality [16, 21]. In addition, despite the fact that games are usually large and complex software projects with high individuality, one can identify a variety of common concepts (e.g., maps, weapons, terrains, etc.), which can enable reuse among games of the same genre. To introduce reuse into the game development process, several studies have proposed software architectures that improve the reusability of games (e.g., [15, 18, 28]). The aim of such architectures is to deliver more stable and extensible software, with enhanced interoperability, robustness and scalability.

In most of the cases, solutions that facilitate reuse discuss the utilization of componentized opportunities (e.g., [12, 34]). In software engineering, components are typically equivalent to software packages or groups of classes that encapsulate a set of related and well defined functions [40]. By taking into account the enormous amount of source code that is available in Open Source Software (OSS) repositories (e.g., Sourceforge, Github, etc.), in this paper we perform an exploratory case study to investigate the opportunity to reuse OSS components in game development. To achieve this goal, we exploit a large-scale repository of OSS components (namely Percerons[1]) that at this point offers approximately 3,000 components retrieved from open source games. The case study aims at investigating the available open source components, which can be supplied for reuse in the game development community, based on:

(p1) ***Game genre specificity***: By taking into account that software reuse is more efficient when performed within the same application domain [24], we investigate how many components have been identified for each game genre (e.g., *sports* games, *strategy* games, *RPGs*, etc.). It is expected that game genres with high availability of components, can more easily benefit from OSS reuse. The game genres that we investigate are extracted from sourceforge.net, i.e., the source code repository, on which the games have been originally published. The studied genres are: *arcade*, *board*, *card*, *first person shooter*, *puzzle*, *role-playing*, *sports* and *strategy* games.

(p2) ***Requirements specificity***: Even within a specific game genre, components can be further classified, based on the requirement that they implement. Such a classification would provide an even more fine-grained level of specificity, based on which we can further quantify the supply of components. For instance, a component that is related to the scenario of a game, e.g., an inventory of a player in an RPG, is only reusable in scenarios that involve the management of objects collected by game characters. To this end, we have manually classified a subset of the components of the *Percerons* database in seven categories: *scenario*, *controls*, *community*, *speed*, *characters*, *sound*, and *graphics*. The categories have been retrieved from the work of Ham et al. [22], on gamers' satisfaction

[1] http://www.percerons.com.

factors. The connection between game satisfaction factors and requirements is discussed in Sect. 2.3.

(p3) ***Reusability***: However, the identification of a software component is only the first step towards its reuse. The next step is its adaptation to the target system. The ease of adapting a software component in a new system is quantified through the *reusability* quality attribute [1]. Therefore, we investigate if there are statistically significant differences in the reusability of components, identified in games of different genres.

The rest of the paper is organized as follows: In Sect. 2 we introduce the concepts of *software reuse* and *component-based software engineering*. Additionally, we provide background information that is used in this study, i.e., aspects of *game engineering* and *the component extraction algorithm* of Percerons. In Sect. 3 we present the study design in the form of a case study protocol. In Sect. 4 we provide the results, organized by research question, and discuss them in Sect. 5. In Sect. 6 we discuss the threats to validity of our study, and in Sect. 6, we conclude the paper.

2 Background Information

2.1 Software Reuse

Software reuse is the process of implementing or updating software systems using existing software assets [26]. Software reuse according to Baldassaire [8] is a software engineering technique that, when adopted systematically, can improve and even guarantee software quality. Additionally, it is suggested that reuse ***has a positive effect on productivity and quality*** [8]. The results of the previous study are verified in [32] where traditional and reuse-based software productions are compared in an industrial context. Furthermore, a failure mode model for part-based software reuse was proposed to improve the reuse processes [16].

Source code ***reuse*** is considered to be more ***intense*** in OSS development compared to commercial/closed source software [31]. Heinemann et al. performed an empirical multiple-case study in 20 popular OSS Java projects and concluded that third party reuse is common in OSS [23], while Raemaekers et al. [36] pointed out that logging frameworks (e.g., log4j) are the most frequently reused libraries. Sojer and Henkel [39] investigated, through a survey among 686 open-source developers, the usage of existing open-source code for the development of new open-source software. Their results showed that on average 30% of the offered functionality is based on reuse.

Another type of studies aims at ***diversifying between white-box and black-box reuse***. According to Heinemann et al. [23] black-box reuse is the predominant form of reuse. These findings are in accordance with those of Haefliger et al. [21], who concluded that black-box reuse is the dominant form of reuse by analyzing six open source projects and interviewing their developers. Schwittek and Eicker [38] examined black-box reuse in OSS web applications resulting that on average this type of applications reuse 70 libraries, 50% of which come from the Apache Foundation. White-box reuse has been studied by Frakes et al. and Mockus et al. on 38.7 thousand OSS projects, by measuring filename overlapping. The results showed that more than

50% of the components are reused in more than one projects [16] and [31]. In general it seems that identifying application domains [38], requirements specificity [36] and type of reuse [16, 23, 31] is of great importance in guiding practitioners on where to find appropriate components of reuse.

2.2 Component-Based Software Engineering

Component-Based Software Engineering (CBSE) is an approach that relies on software reuse. CBSE purpose is twofold: (a) to facilitate the development of reusable components that can be used in various independent systems, apart from the one initially implemented for (i.e. development ***for*** reuse), and (b) to exploit reusable components for the development of new systems (i.e. development ***with*** reuse).

In the literature a variety of terms regarding software components can be found, as the term "component" is considered so generic that is used to denote any software part: architectural, design, source code, or requirements unit [17], patterns or even methods and lines of code [14, 40]. In JavaBeans the component is considered to be a class, in Component Object Model (COM) and CORBA Component Model (CCM) a component is an object, whereas in SOFA, PECOS and Pin it is an architectural unit [27]. However, Szyperski [40] distinguishes between classes and components: components are more abstract than classes and can be considered to be stand-alone service providers consisting of one or more classes. Components are "fired" during execution and therefore considered as deployment units, while classes are considered as development artifacts. Unlike classes, components can be synthesized with different technologies and can contain elements such as global variables, images, html files, etc.

Component adoption in software reuse may occur in many levels of granularity from a few lines of code to even a whole system [2]. Franch et al. point out the importance of the component selection process in software engineering, a fact that indicates the growing need for establishing software reuse patterns and guidelines [17]. The separation of the components' interface from the components' functionality is an important aspect of a component that may increase its reuse. For this reason according to [14] the use of design patterns in components analysis and design can be useful in increasing component cohesion and minimizing component internal coupling.

2.3 Game Engineering

The main requirement of every game is to be entertaining (see [11, 25, 41]) and therefore gamers' satisfaction factors are of paramount importance in the game analysis phase. The first study that investigated the factors from which gamers gain satisfaction was performed by Ham et al. [22]. The results of the study suggested that game satisfaction factors are game genre specific. Ham et al. investigated seven satisfaction factors (Scenario, Graphics, Sound, Game Speed, Game Control, Character and Community) and several game genres (Role Playing Games - RPG, First Person Shooter - FPS, Sport Video Games and Computer-Mediated Board Games). The average importance of each factor, calculated over all game genres, is depicted in Table 1.

Table 1. User satisfaction factors [22]

Id	Factor	Importance	Id	Factor	Importance
1	Character	20,0 %	5	Scenario	11,1 %
2	Graphics	17,6 %	6	Sound	10,8 %
3	Game Control	16,7 %	7	Community	10,1 %
4	Game Speed	13,7 %			

While discussing the results of this paper, we have to note that this study has been published a decade ago, when the state of practice in game industry was substantially different. A replication of the aforementioned study has been published in 2014, by Paschali et al. [33]. In the recent study, the results have been updated: Character Solidness, Scenario and Sound are highlighted as the most important factors for gamers' satisfaction, followed by Game Speed, Game Community, Controls and Graphics. The fact that the results of the two studies are contradicting is considered rather intuitive, in the sense that such factors are highly related to the most popular game genre, and the state of practice in the industry. In this study, we reuse the ***game satisfaction factors*** as ***types of requirements***.

2.4 An Algorithm for Component Identification

In this section we shortly describe the methodology that is used in the study to identify components from open source games, as proposed by Ampatzoglou et al. [4]. The used algorithm is based on the identification of reusable sets of classes, by applying a path-based strong component algorithm [19]. To apply this algorithm a directed graph is created that depicts the dependencies among the classes of the system and then depth-first search is performed to identify strongly connected components, in our case: sets of classes. The algorithm successively provides sets of classes that are as independent as possible, grouped together according to the functionality that they offer. In particular the steps of the applied methodology are the following:

step 1. Create a dynamic two dimensional array where *Candidate Components* will be stored in. Each row will store groups of classes that depend on each other. In row 1 only one class will be stored depending solely on itself. In row two, couples of classes will be stored that depend on each other, in row three triplets of classes will be stored presenting dependencies, etc. Each row number defines the maximum number of classes that can be included in a *Candidate Component*. The columns represent the number of possible *Candidate Components* that can be used for each component size. At this step only the first *Component Candidate*, of size 1, is created for one class of the system.

step 2. Identify the classes that the participants in the *Candidate Components* identified in the previous step are connected to.

step 3. Sort the dependencies according to their number of external dependencies in a descending order.

step 4. For every dependency create an updated *Component Candidate* and place it in the corresponding position in the array according to the number of classes in the dependency group.

step 5. Return to step 2, for every Component Candidate created in the previous step, according to the order that they have been added in the array. The process stops if the maximum number of components is reached or if there are no external dependencies.

step 6. For every dependency in the list create an updated *Component Candidate* and place it in the corresponding position in the data structures.

step 7. For every Component Candidate created in the previous step, following the order that each candidate was identified, return to step 2. Stop if maximum number of components is reached or if there are no external dependencies.

For example, by applying the algorithm on the dependency graph of Fig. 1, we obtain the candidate components presented in Table 2. The intermediate steps on the application of the algorithm are presented in detail in the original study [4]. We note that from the candidate components identified by this algorithm, we only investigate those that are independent of other system classes (i.e., have zero efferent coupling [29]).

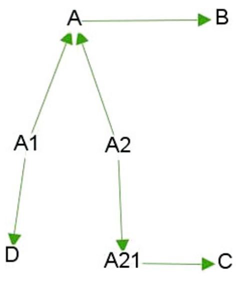

Fig. 1. Dependency graph (Example)

Table 2. Extracted components (Example)

Size 1	A	A1	A2	A21	B	C	D
Size 2	A,B	A1,D	A,A1	A,A2	A2,A21	A21,C	A,C
Size 3	A,A1,D	A,A1,B	A,A2,B	A,A2,A21	A,A21,C	A2,A21,C	A,B,C
Size 4	A,A1,B,D	A,A2, A21,B	A,A2,A21,C				
Size 5	A,A2,A21, B,C						

3 Case Study Design

In this section, we present the protocol that has been used for guiding the execution of this case study. The case study has been designed and is reported based on the guidelines of Runeson et al. [37]. Therefore, in Sect. 3.1 we present the aim of the study and the research questions in which we decompose it, in Sects. 3.2 and 3.3 we describe the case selection and the data collection processes, and in Sect. 3.4, we provide an overview of the data analysis process.

3.1 Research Question

The goal of this case study, based on GQM [10], is to ***characterize*** OSS components ***with respect to*** their domain-specificity and reusability ***from the point of view of*** software engineers ***in the context of*** game development. To ease the design and reporting of the case study, we split the aforementioned goal into three research questions, based on the analysis perspectives (i.e., game genre specificity, requirements type specificity, and reusability) that we introduced in Sect. 1, as follows:

[RQ_1]: *Which game genres offer the most open source components*?
This research question aims at identifying game genres that offer the larger pool of components. The game genres that are used in this study have been extracted from sourceforge.net, i.e., the repository from which the OSS projects have been retrieved. The categorization on sourceforge.net is performed by the game developers, and therefore is considered accurate. The analysis will provide an overall view of how many components are found on average in each game genre.

[RQ_2]: *Which are the game requirements to which most open source components are related*?
This question explores the types of requirements for which the most components are implemented. Requirements are mapped to game satisfaction factors, as presented in Sect. 2.3 (see [33]). The analysis will provide insight on the game requirements for which components are more easily accessible, based on the quantitative analysis.

[RQ_3]: *What is the reusability of open source components for each game genre*?
The two quality attributes related to software reuse are functionality and reusability. These attributes will be analyzed for the components retrieved across different game genres.

[$RQ_{3.1}$]: *Is there a difference in the average functionality offered by open source components for various game genres*?

[$RQ_{3.2}$]: *Is there a difference in the average reusability of open source components for various game genres*?

The results of this research question are expected to provide insights on how easy it is to reuse one component, upon its identification.

3.2 Case Selection

The case study of this paper is a holistic multiple-case study [37] for RQ_1 and an embedded-multiple case study for RQ_2 and RQ_3. The context of the study is OSS game development, the cases are open source games (for RQ_1 games are also the units of analysis), and units of analysis (for RQ_2 and RQ_3) are open source components.

In order to select as many cases as possible for our case study, we exploited a repository of open source components, namely Percerons (see http://www.percerons.com). Percerons is a software engineering platform [5] created by one of the authors with the aim of facilitating empirical research in software engineering, by providing: (a) indications of componentizable parts of source code, (b) quality assessment, and (c) design pattern instances. The platform is consistently used for empirical research in the last three empirical software engineering conferences (ESEM' 13 [6], ESEM'14 [20], and ESEM' 15 [7, 35]). The identification of units of analysis is performed automatically, by dumping the complete database of the repository.

In its current state *Percerons* provides 6.4 million candidate components that concern 8 application domains. From these candidate components, 1.1 million have been retrieved from OSS computer games. However, we need to note that the majority of these components are not completely independent, since the algorithm described in Sect. 2.4 stores components with efferent coupling less than 10. In our case study as units of analysis, we consider approximately 3,000 components that are completely independent and compileable (i.e., efferent coupling equals zero). The average size of the components that are used as units of analysis is 6.52 classes (standard deviation: 8.92), ranging from single class components to components up to 40 classes.

3.3 Data Collection

In order to answer our research questions for every open source game that we analyzed we recorded the following variables:

- ***Game Name***: The name of the open source game that we analyzed.
- ***Game Genre***: The genre of the game—*Arcade*, *Board*, *Card*, *FPS*, *Puzzle*, *RPG*, *Sports* and *Strategy*. We note that some categories that are obtained from Percerons have been excluded or merged, due to the low number of games that they involved. For example, *Educational* games have been removed, *Turn-Based* and *Real-Time* Strategy games have been merged in a common category, named *Strategy*.
- ***Number of Components***: The number of independent and compileable components that have been identified for the current game.

Additionally, for each component the following variables have been recorded:

- ***Component ID***: A unique identifier for the component.
- ***Game Genre***: Derived from the case variables.
- ***Requirement Type:*** The type of requirement that the component implements. The possible classes for this variable are: *Scenario*, *Controls*, *Community*, *Speed*, *Characters*, *Sound*, and *Graphics*. We note that since this was a manual process, it was performed on only a limited number of components. In particular, we explored 100 random components, of various sizes, extracted from different games, belonging to various game genres.
- ***Reusability:*** The reusability, as provided by the Percerons database, is calculated based on the Quality Model for Object-Oriented Design (QMOOD) [9]. QMOOD suggests that reusability is calculated as a function of component size in classes, cohesion, coupling, and public interface. By taking into account: (a) the rigorous empirical validation of QMOOD by experienced software engineers, and (b) its popularity in the software engineering literature, we assume that it is a valid model for quantifying reusability. In any case, we note that at this stage we are not interested in the actual value of reusability, but only on components ranking.

- ***Functionality:*** As a measure of functionality we use Afferent Coupling (AffC), as proposed by Martin [29]. Afferent coupling counts the number of system classes that actually invoke any method of the public interface of the component. In that sense, it is a proxy of the functionality that this component offers to the rest of the system. Thus, a component that provides high functionality to other system classes is more probable to be reused than another that only provides limited services, even in its original system.

3.4 Data Analysis

The data analysis step of this case study includes the calculation of descriptive statistics, and the application of independent sample t-tests and Analysis of Variance (ANOVA). Table 3 summarizes the data analysis process that we have applied in this case study.

In particular for ***RQ₁*** the number of components retrieved per game genre is presented along with basic descriptive statistics (i.e., minimum, maximum, and average number of components per game). Also the standard deviation which is calculated to quantify the amount of variation in the number of components per game is presented. Additionally Analysis of Variance is performed to identify whether there are certain game genres that offer significantly more components. One limitation of ANOVA is the fact that it identifies differences in the mean value of the testing variable, among groups, but it does not specify which groups are different. Therefore, the results of ANOVA are further explored with independent sample t-tests, in order to identify which game genres (i.e., the grouping variable) are different in terms of the number of components they offered (i.e., independent variable).

Concerning ***RQ₂***, we discuss the frequency with which components implement various requirement types. The results are presented in the form of a pie chart. The same descriptive statistics as RQ_1 are presented for reusability and functionality metrics with respect to the various game genres, addressing **RQ₃**. In that case ANOVA and independent samples t-test are performed to identify whether different game genres offer components that present *significant* differences in reusability and functionality.

Table 3. Data analysis and presentation overview

RQ	Variable	Analysis
Components / Genre	Number of Components *Grouping Variable*: Game Genre	• Descriptive statistics (mean, min, max, std. dev.) • Frequencies • ANOVA
Components / Requirements	Number of Components *Grouping Variable*: Requirement Type	• Frequencies (pie chart)
Reusability / Genre	Reusability Functionality *Grouping Variable*: Game Genre	• Descriptive statistics (mean, min, max, std. dev.) • Frequencies • ANOVA

4 Results

In this section we present the results of our case study, organized by research question, and based on the data analysis plan, as presented in Sect. 3.4. Therefore, first we present the results as obtained by the statistical analysis and then interpret them.

RQ1 (Availability of Components for Game Genres). Table 4 presents the results that have been obtained by splitting the dataset by game genre and then calculating basic descriptive statistics. The results of Table 4 are ranked by the mean value of components offered by one game (see column 4). It can be observed that the game genre that has the highest number of components (see Frequency—column 3) is *Board* games, followed by *Puzzles*. However, we need to underline that these game genres are the ones with the most games in the dataset (see N—column 2). In terms of average components per game, we observe that the maximum value exists for *FPS* and *Strategy* games, whereas the least components per game are found in *Board, Card* and *Puzzle* games. Thus, based on this ranking we can claim that the amount of components that are available for Board and Puzzle games are only due to the number of explored games, and not due to game-specific characteristics.

To investigate if the aforementioned differences are statistically significant, we first perform an Analysis of Variance (ANOVA), which suggested that some of the game genres offers significantly more components per game (F: 3.62, sig: 0.00). Next, in order to identify which game genres are those that stand out, either positively or negatively, we performed independent sample t-tests. The results revealed that the top-2 genres (i.e., *FPS* and *Strategy* games) are indeed having more available components than the rest game genres. The second group of game genres (i.e., *RPG* and *Sport* games), although offer on average approximately 10 additional components compared to the other genres, this result is not statistically significant.

A possible explanation of the aforementioned ranking is the level of game logic complexity of every game genre. For example, *Arcade*, *Puzzle*, *Card* and *Board* games have a rather limited game logic (at least compared to the other genres), less impressive graphics, etc. Therefore, the amount of possible components is limited. On the other hand, the various characters, scenario objects, etc. offered in FPS, Strategy, Sports games and RPGs, offer many reuse opportunities.

Table 4. Component per game genre

Genre	N	Frequency	Mean	Std. Dev	Min	Max
First Person Shooter (FPS)	8	400	50.00	36.02	3	99
Strategy	9	438	48.67	23.71	17	83
Sports	6	212	35.33	27.48	7	72
RPG	10	348	34.80	26.34	9	76
Arcade	17	407	23.94	12.19	8	45
Puzzle	21	464	22.10	18.39	1	64
Card	7	153	21.86	18.89	5	59
Board	31	647	20.87	18.49	4	80

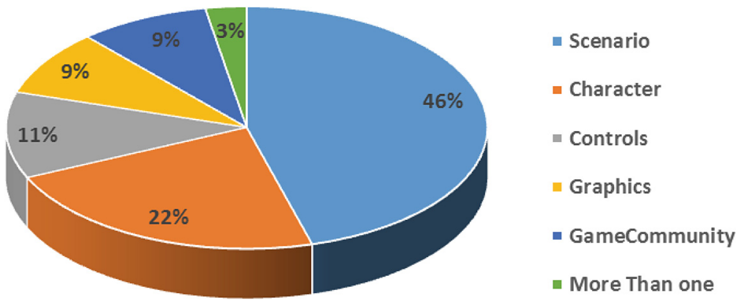

Fig. 2. Pie Chart (Frequency of Requirement types)

RQ2 (Availability of Components for Requirement Types). Concerning RQ_2, we discuss the frequency with which components implement the various requirement types (see Fig. 2). The results of the pie chart suggest that most of the identified components are implementing requirements that concern the game *Scenarios*, followed by *Characters*. Another interesting finding is that we were not able to identify any component that is related to game *Speed*[2].

The fact that game speed has not been associated with any component is intuitive in the sense that speed is a run-time characteristic that cannot be identified with static source code analysis. In addition, the extensive linkage of components to scenarios and characters is in accordance to our discussion for RQ_1 suggesting that most of the components are found in games with complex game logic.

RQ3 (Reusability of Components for Game Genres). In order to investigate the reusability of components that are extracted from different game genres, we performed descriptive statistics, ANOVA, and independent sample t-tests for two testing variables: component functionality (afferent coupling) and component reusability. In Table 5, we present descriptive statistics concerning the afferent coupling of components extracted from different game genres. The results suggest that *RPGs*, *FPSs*, and *Sport* games offer components that are more intensively used inside their games. This fact can be explained by the average size of these games, in the sense that games with more classes are expected to have more method invocations to the extracted components. Another interesting finding is that all differences that are presented in Table 5 are statistically significant and therefore generalizable to the population, according to the individual independent sample t-tests. As expected, ANOVA has also revealed a difference between the groups (F: 46.18, sig: 0.00).

Similarly in Table 6, we present the results on the reusability of components extracted from different game genres. The descriptive statistics imply that differences between games genres are rather small in absolute numbers with the only exception of

[2] A very small number of classes has been related to sound requirements, but due to its negligible number has not been included in the pie chart.

Table 5. Component functionality per game genre

Genre	N	Mean	Std. Dev.	Min	Max
Arcade	407	11.76	13.00	0	61
Board	647	19.70	24.48	0	109
Card	153	28.83	41.70	0	207
First Person Shooter (FPS)	400	38.72	49.84	0	234
Puzzle	464	15.54	19.33	0	70
RPG	348	43.69	86.97	0	337
Sports	212	33.62	39.14	0	148
Strategy	438	24.12	35.97	0	152

Table 6. Reusability per game genre

Genre	N	Mean	Std. Dev.	Min	Max
Arcade	407	3.313	2.433	0.375	15.633
Board	647	3.576	2.525	0.250	22.516
Card	153	3.623	2.741	0.333	24.025
First Person Shooter (FPS)	400	4.328	4.039	-0.385	69.250
Puzzle	464	3.685	2.868	0.119	18.517
RPG	348	3.768	2.603	0.500	17.034
Sports	212	3.681	4.081	0.308	66.552
Strategy	438	3.550	2.727	0.500	20.026

FPS games. Additionally, although the results of ANOVA (F: 10.11, sig: 0.00) suggest the existence of significant differences, the independent sample t-tests revealed that these are limited to the difference of FPSs with all other game genres. The outcome of the statistical analysis suggests that differences in the reusability of open source games are rather small, regardless of game genre.

5 Discussion

The results of this paper revealed that the top-2 genres ***FPS and Strategy*** games *offer significantly more components* than the rest game genres. In terms of *requirements specificity*, most of the identified components are implementing requirements that concern the game ***Scenarios***, followed by ***Characters***. Concerning *component functionality* ***RPGs***, ***FPSs***, and ***Sport*** games offer components that are more intensively used inside their games, while in terms of *component reusability* no significant differences between games genres are found with the only exception of ***FPS*** games. The results of this study provide useful information both to researchers and practitioners:

- ***Guidance on the existence of reuse opportunities for practitioners***. Based on the results of this study, game developers can have indications on the feasibility of reuse in different game genres.

  – *FPS game developers can exploit the great reuse opportunities offered by OSS components*. This application domain offers the most components per game that offer substantial functionality inside games, and are of optimum design-time reusability.
  – *Strategy, Sport and Role-Playing game developers can also exploit the large number of components offered by OSS games*, although they have some limitations. For example, *RPGs* offer the most functional components, of high structural reusability. However, their availability is lower than that of FPS games. On the other hand, despite the fact that *Sport* games that offer a high number of components, these components are not of optimal reusability or functionality.
  – Game developers of any game genre should *consider reuse* of OSS components *when implementing requirements related to scenarios and character management*.
- ***Guidance on case selection for researchers***. Nowadays, more and more researchers perform empirical studies on OSS projects. The results of the study can guide researchers in selecting appropriate game genres to identify as many cases/units of analysis as possible.
- ***Future work opportunities for researchers.*** Some interesting future work directions are derived from this study: (a) the actual reuse rates of these components in OSS games can be calculated, (b) the reusability of these components can be tested by software engineers through experiments, and (c) a process for systematically reusing these components can be introduced.

6 Threats to Validity

In this section we discuss threats to the validity of our case study, with regard to construct, reliability and external aspects [37]. Threats to internal validity are not discussed in this paper, since identifying causal relations was out of the scope of this study. A possible threat to ***construct validity*** is related to the metrics that are used to answer our research questions and the extracted components. In particular, we have used QMOOD to measure reusability and Afferent Coupling (AffC) to measure functionality. Although we acknowledge that if different measures are used, the results might be slightly altered, we believe that both choices provide adequat assessments of the corresponding quality attributes. QMOOD, is an established quality model that has been rigorously validated [9], whereas AffC offers a well-known proxy of functionality, as explained in Sect. 3.3. Finally, another threat to construct validity is whether the candidate components are indeed reusable artifacts that can be ported to settings beyond their own game. We believe that the component selection algorithm, which is based on an exhaustive search process, provides adequate recall rates, and therefore is fitting for the purposes of this study. In any case to the best of our knowledge there is no algorithm that 100% accurately captures all intended components of the original developers.

With regard to ***reliability***, we consider any possible researchers' bias, during the data collection and data analysis process. In particular in the data collection phase, the only possible bias can be identified in RQ_2. To gather data on the types of requirements that components implement we employed a manual process performed by the first author. In order to increase the reliability of this process the second and the third author validated the results. Finally, concerning ***external validity***, a potential threat to generalization is that if the component extraction algorithm was performed on additional, or different games, the results might be altered. However we believe that the selected cases (open source games), offer a large and representative sample of the population. Additionally, we need to clarify that although, the small amount of cases for RQ_3 is a threat to generalization, the manual inspection of additional games was not possible due to the time consuming nature of the manual inspection.

7 Conclusion

In this paper, we empirically explore an important topic in game development, i.e., the opportunity to reuse components from existing games. As parameters in this empirical study we selected two aspects that can affect reusability: the application sub-domain of the game, namely the *game genre*, and the *requirement specificity* that a certain component may fulfill. To evaluate the relation of the game genre and the requirement types in games components, approximately 3,000 components were retrieved from over 100 open source games. The results of the study suggested that specific game genres offer more reuse opportunities than others, and that most components are related to scenario and characters. Based on these results, we have been able to provide useful implications for researchers and practitioners. As future work, we plan to replicate the study with more refined metrics/algorithms and feedback from game developers. Additionally, we plan to perform an in-depth study of a small number of games where the actual components that were envisioned for reuse are actually used for this purpose.

References

1. 9126-2001: ISO/IEC, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland (2001)
2. Ajila, S.A., Wu, D.: Empirical study of the effects of open source adoption on software development economics. J. Syst. Softw. Elsevier **80**(9), 1517–1529 (2007)
3. Ampatzoglou, A., Stamelos, I.: Software engineering research for computer games: A systematic review. Inf. Softw. Technol. Elsevier **52**(9), 888–901 (2010)
4. Ampatzoglou, A., Stamelos, I., Gkortzis, A., Deligiannis, I.: Methodology on extracting reusable software candidate components from open source games. In: Proceeding of the 16th International Academic MindTrek Conference, pp. 93–100. ACM, Finland (2012)
5. Ampatzoglou, A., Michou, O., Stamelos, I.: Building and mining a repository of design pattern instances: Practical and research benefits. Entertainment Comput. Elsevier **4**(2), 131–142 (2013)

6. Ampatzoglou, A., Gkortzis, A., Charalampidou, S., Avgeriou, P.: An embedded multiple-case study on oss design quality assessment across domains. In: 7th International Symposium on Empirical Software Engineering and Measurement (ESEM 2013), pp. 255–258. ACM/IEEE Computer Society, Baltimore, USA, 10–11 October 2013
7. Arvanitou, E.M., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: Introducing a ripple effect measure: a theoretical and empirical validation. In: 9th International Symposium on Empirical Software Engineering and Measurement (ESEM 2015), ACM/IEEE Computer Society, Beijing, China
8. Baldassarre, M.T., Bianchi, A., Caivano, D., Visaggio, G.: An industrial case study on reuse oriented development. In: 21st International Conference on Software Maintenance (ICSM 2005), IEEE Computer Society, 283–292, September 2005
9. Bansiya, J., Davies, C.G.: A hierarchical model for object-oriented design quality assessment. Trans. Softw. Eng. IEEE Comput. Soc. **28**(1), 4–17 (2002)
10. Basili, V.R., Caldiera, G., Rombach, H.D.: Goal question metric paradigm, Encyclopedia of Software Engineering, pp. 528–532. John Wiley & Sons, New York (1994)
11. Callele, D., Neufeld, E., Schneider, K.: Emotional requirements in video games. In: 14th International Conference on Requirements Engineering, IEEE Computer Society, Minneapolis, USA,11 – 15 September 2006
12. Cho, H., Yang, J.S.: Architecture patterns for mobile games product lines. In: Proceedings of the 2008 International Conference on Advanced Communication Technology (ICACT 2008), pp. 118–122. IEEE Computer Society Korea, 17 – 20 February 2008
13. Consumer Electronics Association, "Digital America", published electronically at. http://www.ce.org
14. Crnkovic, I., Hnich, B., Johnson, T., Kiziltan, Z.: Specification, implementation, and deployment of components. Commun. Assoc. Comput. Mach. **45**(10), 35–40 (2002)
15. Folmer, E.: Component based game development – a solution to escalating costs and expanding deadlines? In: Schmidt, H.W., Crnković, I., Heineman, G.T., Stafford, J.A. (eds.) CBSE 2007. LNCS, vol. 4608, pp. 66–73. Springer, Heidelberg (2007)
16. Frakes, W.B., Fox, C.J.: Quality improvement using a software reuse failure modes model. Trans. Softw. Eng. IEEE Comput. Soc. **22**(4), 274–279 (1996)
17. Franch, X., Carvallo, J.P.: Using quality models in software package selection. Softw. IEEE Comput. Soc. **20**(1), 34–41 (2003)
18. Furini, M.: An architecture to easily produce adventure and movie games for the mobile scenario. Comput. Entertainment Assoc. Comput. Mach. **6**(2), 1–16 (2008)
19. Gabow, H.N.: Path-based depth-first search for strong and bi-connected components. Inf. Process. Lett. Elsevier **74**(3–4), 107–114 (2000)
20. Griffith, I., Izurieta, C.: Design pattern decay: the case for class grime. In: 8th International Symposium on Empirical Software Engineering and Measurement (ESEM 2014), ACM/IEEE Computer Society, Torino, Italy, 18–19 September 2014
21. Haefliger, S., von Krogh, G., Spaeth, S.: Code reuse in open source software. Manage. Sci. PubsOnline **54**(1), 180–193 (2007)
22. Ham, H., Lee, Y.: An empirical study for quantitative evaluation of game satisfaction. In: 2006 International Conference on Hybrid Information Technology, pp. 724–729. ACM, November 2006
23. Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., Irlbeck, M.: On the extent and nature of software reuse in open source java projects. In: Schmid, K. (ed.) ICSR 2011. LNCS, vol. 6727, pp. 207–222. Springer, Heidelberg (2011)
24. Johnson, I., Snook, C., Edmunds, A., Butler, M.: Rigorous development of reusable, domain-specific components, for complex applications. In: 3rd International Workshop on Critical Systems Development with UML (CSDUML 2004), Springer (2004)

25. Kasurinen, J., Maglyas, A., Smolander, K.: Is requirements engineering useless in game development? In: Salinesi, C., Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 1–16. Springer, Heidelberg (2014)
26. Krueger, C.W.: Software reuse. Comput. Surv. ACM **24**(2), 131–184 (1992)
27. Lau, K.K., Wang, Z.: A taxonomy of software component models. In: 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA), pp. 88–95. IEEE (2005)
28. Lee, W.P., Liu, L.J., Chiou, J.A.: A component-based framework to rapidly prototype online chess games for home entertainment. In: Proceedings of the International Conference on Systems, Man and Cybermetrics (SMC 2006), IEEE Computer Society, Taipei, Taiwan, pp. 4011–4016, 8–11 October 2006
29. Martin, R.C.: Agile software development: principles, patterns and practices. Prentice Hall, New Jersey (2003)
30. McShaffry, M.: Game Coding Complete. Paraglyph Press, Arizona, USA (2003)
31. Mockus, A.: Large-scale code reuse in open source software. In: 1st International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS 2007), IEEE Computer Society (2007)
32. Morisio, M., Romano, D., Stamelos, I.: Quality, productivity, and learning in framework-based development: an exploratory case study. Trans. Softw. Eng. IEEE Comput. Soc. **28**(9), 876–888 (2002)
33. Paschali, M.E., Ampatzoglou, A., Chatzigeorgiou, A., Stamelos, I.: Non-functional requirements that influence gaming experience: A survey on gamers satisfaction factors. In: 18th Academic MindTREK Conference (MindTREK 2015), ACM, 4–6 November 2014, Tampere, Finland
34. Passos, E.B., Weslley, J., Walter, E., Clua, G., Montenegro, A., Murta, L.: Smart composition of game objects using dependency injection. Comput. Entertainment, Assoc. Comput. Mach. **7**(4), 408–423 (2009)
35. Reimanis, D.: A research plan to characterize, evaluate, and predict the impacts of behavioral decay in design patterns. In: 13th International Doctoral Symposium on Empirical Software Engineering (IDOSE 2015), Beijing, China
36. Raemaekers, S., van Deursen, A., Visser, J.: An analysis of dependence on third-party libraries in open source and proprietary systems. In: 6th International Workshop on Software Quality and Maintainability (SQM 2012), March 2012
37. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons, Hoboken (2012)
38. Schwittek, W., Eicker, S.: A study on third party component reuse in java enterprise open source software. In: 16th International Symposium on Component-based Software Engineering (CBSE 2013), pp. 75–80. ACM (2013)
39. Sojer, M., Henkel, J.: Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. J. Assoc. Inf. Syst. **11**(12), 868–901 (2010)
40. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley International, Massachusetts, USA (1997)
41. van Lent, M., Swartout, W.: Games: Once more, with Feeling. Comput. IEEE Comput. Soc. **40**(8), 98–100 (2007)