Near-miss Software Clones in Open Source Games: An Empirical Study

Yaowen Chen Iman Keivanloo[+] Chanchal K. Roy
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
yac508@mail.usask.ca, croy@cs.usask.ca
[+]Department of Electrical and Computer Engineering
Queen's University
Kingston, Canada
iman.keivanloo@queensu.ca

***Abstract*—Developers tend to reuse source code by copy/paste. This form of reuse introduces code clones to software systems. Cloning in games can happen in different levels of granularity. The extreme case is known as Game Clone where the complete project is being cloned, e.g., by making a new independent branch which the original game's source code constitutes the seed for the new branch. Although it has been more than two decades since research on code clones started, various characteristics of cloning in open source games has not been studied. Therefore, there is no specific evidence on status of cloning e.g., dominant clone type in games. In this paper, we present an empirical study on code cloning in open source games by applying a state of the art clone detector, NiCad, and a clone visualization and analysis tool, VisCad. We identify both exact and near-miss clones from more than twenty open source C, Java and Python-based games, such as Jake2, Duo, Hexen2, jChess and OpenRPG from five different categories. Furthermore, we analyze a set of metrics for code clones in several different dimensions, including language, category, clone density and clone location to answer six essential research questions about the current status of cloning in open source games. Our research illustrates that cloning happens not only at inter-project level but also intra-project, specifically in First Person Shooter games developed using C (Type-1 50% clones). Such observation concretely shows the necessity of adopting clone management systems for game development.**

I. Introduction

In software development, *code cloning* or duplicate code refers to fragments of source code that occur more than once either within or across different systems [1, 2]. Studies show that considerable amount of code of a software system is cloned code [1]. Whether code cloning is harmful or not is an open question [3]. Nevertheless, it is considered as a potential threat in software maintenance and development [4]. For example, a study from Juergens et al. [4] shows that inconsistent updates on the duplicated code often causes incorrect program behavior.

In response, a large number of code clone detection tools have been proposed [1]. These tools are built to fulfill the demand of different users and stakeholders. Earlier studies focus on the clone detection techniques or tools, and validation or comparison of those techniques [5]. Recently, it is shown that empirical studies on code clones can reveal interesting facts about software development behaviors [2].

In this paper, we provide an in-depth empirical study of both exact and near-miss code clones in open source games. Our motivation is to observe whether game developers tend to make reuse code by cloning. We analyze twenty four open source games written in C, Java and Python-based from five different categories by applying a hybrid code clone detecting tool (NiCad [6]). We manually verify the detected clones and apply VisCad [7] to visualize the results. In particular, we answer six specific research questions as follows:

- *RQ1 - What is the status of code clone in open source games for different languages?*
- *RQ2 - Is there any difference in code clones between the different categories such as First Person Shooters games and Role-Playing games?*
- *RQ3 – Whether cloning affects the majority of game systems?*
- *RQ4 - What are the profiles of cloning density?*
- *RQ5 - Whether intra or inter-game code cloning occurs?*
- *RQ6 - Is there any duplicated code that could be reusable for future game development, such as extracting duplicated code to build libraries?*

The rest of the paper is organized as follows. In Section 2, we present the experimental setup including the selected metrics. Section 3 provides the experimental study, analysis and results. Furthermore, we discuss related work and the threats to validity, following the conclusion in Sections 4, 5, and 6 respectively.

II. Study Setup

In this study, we used NiCad clone detector to find code clones from the subject open source games and game engines. Furthermore, we applied VisCad to calculate the clone related code metrics using the output of NiCad.

Since definition of similarity for code clone is application dependent, we defined the similarity of source code by computing the size-sensitive Unique Percentage of Items (UPI) [2] using Equation 1. We use configuration which is recommended by the tool maintainer for Type-1, 2 (blind renaming), and 3 (blind renaming plus 0.3 UPI threshold which allows up to 30% dissimilarity) and minimum of 5 lines.

$$UPI = \frac{(Number\ of\ Unique\ Items)}{(Total\ Number\ of\ Items)} \quad (1)$$

A. Subject Systems

In this study, we analyze twenty four open source games from five different categories, including four open source Java-based 3D Game Engines, four Java-based Chess Games, four Java-based Role-Playing Games, five Python-based Card Games and seven C-based First Person Shooting Games. The size of subject systems varies from 1K LOC to 844K LOC. Table 1 lists the subject systems with some basic profiles.

TABLE I. THE SUBJECT SYSTEMS

Language	Category	Game	LOC	Total File (TF)
C	First Person Shooter	Chocolate Doom [8]	70000	105
		Doom64 EX [9]	84831	120
		Hexen2 [10]	239531	296
		Open Arena [11]	359981	461
		Smokin' Guns [12]	322403	341
		Unvanguished [13]	844586	1192
		World of Padman [14]	327135	409
Java	Chess	jChecs [15]	23653	102
		jChess [16]	8932	34
		Jose [17]	145548	475
		Mediocre Chess [18]	4084	21
	3D Engine	Ardor3d [19]	79492	413
		Env3d [20]	13054	75
		Jake2 [21]	126026	252
		JMonkeyEngine [22]	244962	1139
	Role Playing Game	Hale [23]	97083	437
		jClassicRPG [24]	102870	685
		OpenRPG [25]	12214	107
		TowerofZaldagor [26]	74011	197
Python	Card Game	Duo [27]	7075	24
		PySolFC [28]	1366	213
		Sabacc [29]	4807	26
		ScoPy [30]	3740	19
		Wizards Magic [31]	6994	48

B. Selected metrics

For this study, we have selected and reused a series of metrics for code clones from earlier studies [2, 7, 32].

Total Cloned Lines of Code (TCLOC). We define the TCLOC as the total number of cloned lines of code in the game for a specific clone type. Additionally, the PTCLOC (Equation 2) refers to the percentage of the total number of cloned lines of code over the total lines of code (TLOC),

$$PTCLOC = \frac{TCLOC}{TLOC} \times 100 \quad (2)$$

PTCLOC provides a realistic view about the cloned code specifically in case of format unification (a.k.a., pretty-printing) during detection. Therefore, in this study, we consider PTCLOC.

File Associated with Clones (FAWC). While TCLOC and the related metrics provide an overall cloning statistics for the subject games, we are interested in metrics to analyze the location of the clone code. FAWC, which is the total number of files that contain one or more cloned fragments for a given clone type, could be used to determine whether clones are local to some specific files or are scattered all over the files of the subject system. It provides a general idea about the location of the clones at the file level. Similar to PTCLOC, we define PFAWC (Equation 3) as the percentage of the file associated with clones over the total number of files in the subject game.

$$PFAWC = \frac{FAWC}{Total\ number\ of\ files} \times 100 \quad (3)$$

A game with high PFAWC means most files in the game are containing code clones. For example, if PFAWC of a game system is 80%, it means 80% of the files containing at least one code clone. In this case, management and reduction of clones are challenging since the clones are distributed in reasonably large number of files.

Cloned Ratio of File for Lines (CRFL). In order to examine the property of code clones in a specific file, we introduce the CRFL into our analysis. Unlike the above metrics, CRFL discovers the files with majority of code clones. In particular, the CRFL (Equation 4) for a specific file *f*, CRFL(*f*), means the ratio of the total number of cloned lines over the total number of lines of code in *f*.

$$CRFL(f) = \frac{Total\ number\ of\ cloned\ line\ in\ file\ f}{Total\ LOC\ in\ file\ f} \times 100 \quad (4)$$

Using CRFL, we can determine whether a file is highly affected by cloning behavior. Eventually, it helps to predict the difficulty of the software maintenance activities. For example, let us say there are two game systems, *x* and *y* with similar TCLOC, but in game *x*, clones are more concentrated in some files with higher CRFL than *y*. It indicates since clones in game *x* are in fewer files than *y*, the game *x* becomes more manageable from clone maintenance perspective.

Clone-cohesion (CCH). For determining the relationships among files or directories from the cloning point of view, the CCH is defined. CCH (Equation 5) measures to what extent the code fragments in a file or directory have cloning relationship with other code fragments within the same file or directory. A file or directory could be defined as highly clone-cohesive, when most of the members of the associated clone class are located inside the concerned file or directory.

$$CCH = \frac{\sum_{i=1}^{n} \frac{Total\ number\ of\ local\ members\ in\ clone\ class\ i}{Total\ number\ of\ members\ in\ clone\ class\ i}}{\#clone\ classes} \quad (5)$$

In CCH, *n* denotes the number of clone classes associated with the target file or directory in the game system, and "local member" means those members of the associated clone class located in the same file or directory. In software maintenance and development, a high CCH is desirable for a system,

because it possibly means the code clones are created for reusing an implementation of similar functionality.

Clone-coupling (CCP). CCP (Equation 6) measures to what extent the code fragments in a file or a directory have clone relationship with other code fragments in different files or directories. In CCP, n denotes the number of the clone classes associated with the target file or directory in the games, and the "foreign members" refer to those members of the associated clone classes that are in different file or directory. For example, a high CCP means that the members of the associated clone classes have clone-pair relationships with many distinct clone classes scattered in the same file or directory. Unlike to CCH, a lower CCP of a game is desirable in our study as such characteristic minimizes the effort to understand subsystem functionality in dealing with the local clones.

$$CCP = \frac{\sum_{i=1}^{n} \frac{Total\ number\ of\ foreign\ members\ in\ clone\ class\ i}{Total\ number\ of\ members\ in\ clone\ class\ i}}{Total\ number\ of\ clone\ classes} \quad (6)$$

III. RESULTS

In this section, we provide the results of our empirical study to answer the identified six research questions using the clone metrics discussed earlier (Section 2-B).

A. RQ1 – Whether cloning happens in games?

In this subsection, we represent the overall level of code clones for different languages, in terms of PTCLOC. Figure 1 summarizes the results of the PTCLOC for different languages: C, Java and Python.

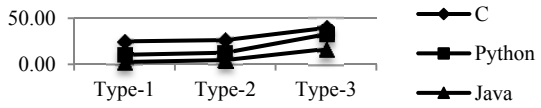

Fig. 1. PTCLOC by languages

First, we observe that there is considerably more cloned code in the C-based open source games, followed by Python-based open source games and Java-based open source games. Specifically, the C-based games contain (Figure 1) a considerable amount exact code clones (Type-1 clone) with 24.89% lines of code duplicated exactly. Exact duplication (Type-1 clone) is much less in the other languages, e.g., 4.59% and 10.37% in Java and Python-based games respectively. Additionally, the results show lack of Type-2 Clones in those games since PTCLOC for all games increases only up to 2% when Type-2 is being considered. Note, Type-2 clones are related to similar code fragment with only different token names such as different variable names. Interestingly, the growing of PTCLOC for all games is much faster for Type-3. PTCLOC of Python-based games increases from 10% (Type-1) to 32.64% (Type-3). Also, PTCLOC of C-based games increases to the highest, 39.55% (Type-3). Although the Java games have the lowest PTCLOC, it still increases up to 16.49% for Type-3 clones. Specifically, Figures 2, 3, 4, 5 and 6 refine Figure 1 to show detailed view of the PTCLOC for the individual open source C-based, Java-based and Python-based games respectively.

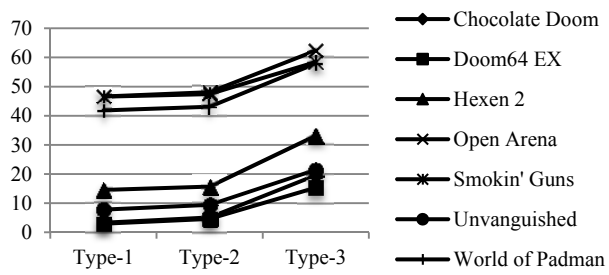

Fig. 2. PTCLOC for C-based games (First Person Shooting Games)

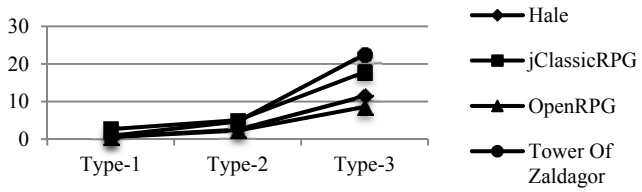

Fig. 3. TCLOC for Java-based games (Role-Playing Games)

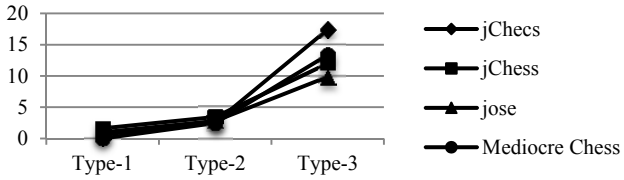

Fig. 4. PTCLOC for Java-based games (Chess Games)

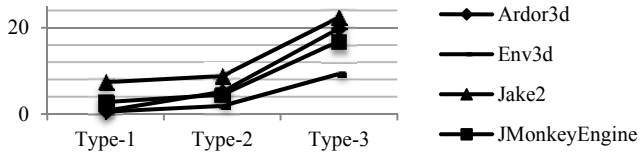

Fig. 5. PTCLOC for Java-based games (3D Game Engines)

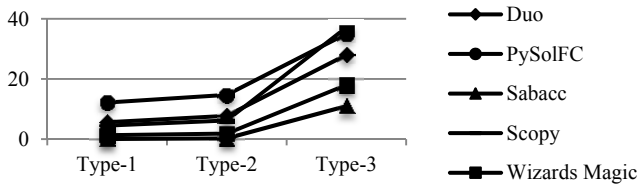

Fig. 6. PTCLC for Python-base games (Card Games)

Figure 2 illustrates a remarkable observation, since unlike Java-based and Python-based games, the PTCLOC of the C-based games vary considerably. Three of the C games, including Open Arena, Smkin' Guns and World of Padman, are holding a surprisingly high level of exact clone (Type-1) with ~50% PTCLOC, whereas the value of others C-based games are under 15%, especially for Doom64 EX, where PTCLOC is only 2.93% for Type-1 clones. Also, for those Java-based games, most of them are remarkably consistent from the PTCLOC metric point of view. Almost all of them have a low level of Type-1 clone (under 5%) and Type-2 (around 5%), and relatively acceptable level of Type-3 (8.6% to 22.6%). Although Jake2 has a higher PTCLOC comparing to the other Java games, its value is still relatively lower than C-based

games. Additionally, in the Figure 6, we notice that the PTCLOC of the Python-based games vary, the Sabacc even has no exact clone (Type-1 Clone), while PySolFC has more than 12% exact clone, which is even larger than some C-based games, such as Chocolate Doom and Doom64 EX. The last observation made in this study is that interestingly, PTCLOC grows considerably fast (~9 times) for Type-3 than Type-2 Clones.

Therefore, for the RQ1, we may conclude that: *cloning happens in open source games. Moreover, cloning frequency depends on programming languages. Specifically C-based games have the highest level of cloning, following by Python-based games and Java-based games. Our study concretely shows the necessity of adopting clone management systems for game development if clones are considered as bad smells.*

B. RQ2 - Is there any difference in code clones between the different categories of games such as First Person Shooters Games and Role-Playing Games?

After studying the effect of programming languages on code clones for games, we analyze how categories impact code clones. We study five specific categories (Table 1) in this paper which are First Person Shooter Game, Card Game, 3D Game Engine, Role-Playing Game and Chess. Figure 7 summarizes the PTCLOC for chosen genres of games.

As Figure 7 illustrates, basically, the PTCLOC of different categories are consistent to their languages. First Person Shooter Games – C-based games – still are the higher in all three types of clones, which have the highest PTCLOC. Moreover, among the Java-based games, Chess games keep the lowest level of code clones, from 0.91% of Type-1 Clones to 11.10% of Type-3 Clones, whereas 3D Game Engines and Role-Playing Games achieve slightly higher value (still very low) where the Card games are placed in the middle.

As a result of this investigation, we answer RQ2 as follows: *basically, the characteristics of code clones in different categories are consistent with the corresponding programming languages. First Person Shooter Games have the highest PTCLOC, following by Card Games, 3D Game Engine, Role-Playing Games and Chess Games.*

C. RQ3 – Are clones scattered over the game systems?

In this subsection, we analyze the proportion of the files in games in association with at least one cloned fragment by using the FAWC and PFAWC metrics. Figure 8 represents the values of PFAWC by languages with different types of clones. First, we notice Python-based and C-based games both hold a high PFAWC (except Java-based games). 50.31% of the files in the C-based games and 42.42% of files in the Python-based games contain at least one exact clone (Type-1), whereas only 12.29% of files in the Java-based games are associated with Type-1 Clone. Additionally, although the C-based games still have most files containing the Type-2 Clone (56.98%), for Type-3 clone, we observe the PFAWC of Python-based games growing fast to be the highest, which is 74.23%, whereas the PFAWC is 69.56% and 54.18% in C-based games and Java-based games. Another notable fact about Type-3 Clones is that there are 54.18% files in Java-based games having cloned code. However, the PFAWC is only 27.31% for Type-2 Clones. It means the Java PFAWC increases about 27%, which is the fast growing ratio for Type-3 clones.

Figures 9, 10, 11, 12, and 13 provide more detailed views of PFAWC for the selected games. First, PFAWC of Open Arena, Smokin' Guns and World of Padman are considerably high (Figure 9) and stable for all types of clones (around 80%). By manually checking the source code, we find that most of those files containing cloned fragments are part of some open source libraries. Those libraries are widely used in games development. Second, PFAWC grows rapidly (Figure 10, 11, 12) in Java-based games as the clone type reaches to Type-3. An example is Hale, which is a Role-Playing Game. There are only 7.3% of files having exact clone (Type-1), but the number increases to 34.3% and 70.5% for Type-2 and 3. Third, Figure 13 illustrates an interesting observation regarding instability of PFAWC for Python games. The extreme cases are Sabacc and PySolFC games. The former has no Type-1 Clone and a few Type-2 Clones in few files, but with more than 55% files having clones. On the other hand, PySolFC begins with a high PFAWC for Type-1 and increases relatively slowly, although it ends with more than 80% PFAWC.

Therefore, for the RQ3, we conclude as: *clones tend to be scattered over C-based games and Python-based games; especially for Type-3 Clones, Python-based games has the highest PFAWC (73%). Moreover, although Java-based games have relatively low PFAWC, the growth of the PFAWC of Java-based games is quite fast as clone type approaches to Type-3*.

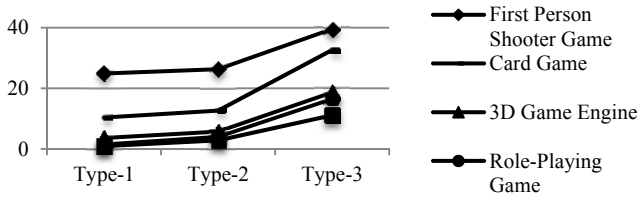

Fig. 7. PTCLOC by categories

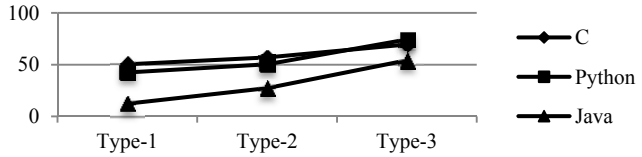

Fig. 8. PFAWC by languages

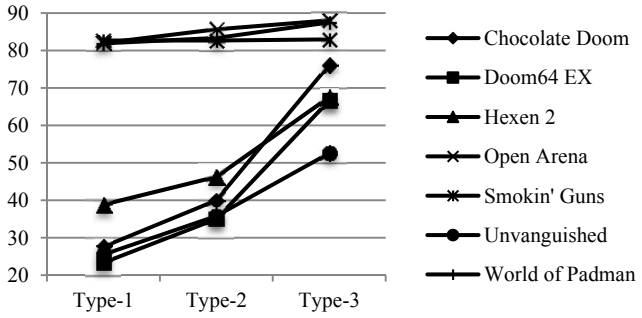

Fig. 9. PFAWC for C-based Games (First Person Shooting Games)

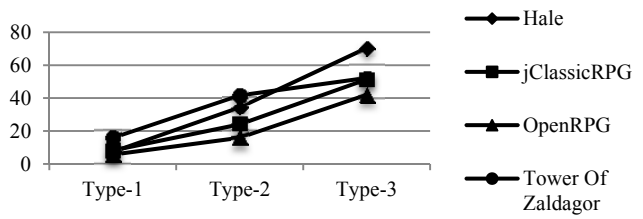

Fig. 10. PFAWC for Java-based games (Role-Playing Games)

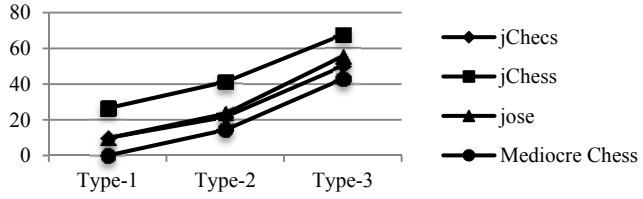

Fig. 11. PFAWC for Java-based games (Chess Games)

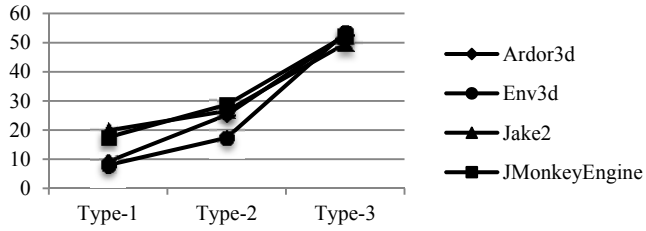

Fig. 12. PFAWC for Java-based games (3D Games Engines)

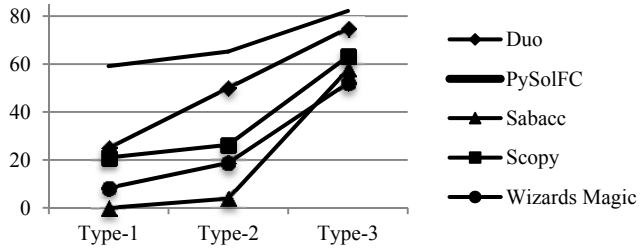

Fig. 13. PFAWC for Python-based Games (Card Games)

D. RQ4 - What are the profiles of cloning density?

Although PFAWC (RQ3) provides an overall view of the location of code clones in the game systems, still more specific information about each file is required to be determined such as which file contains the majority of clones. The CRFL metric family could help us to study the cloning density. Table 2 provides these metrics for the C, Java, and Python-based games.

The average CRFL of the Java-based games is low with 10% for Type-1 Clone (excepting the Jake2, which is 22%), 15% and 2% for Type-2 and Type-3 respectively. In fact, we observed that the CRFL values of most files are low, which means there are only few lines of code being cloned within each file. Considering the PFAWC fast growth of Java-based games (Figure 8), we may conclude that most of the files in the Java-based games contain few cloned lines of code. On the other hand, in the C-based games, Chocolate Doom and Doom64 EX have a low CRFL (versus Java-based games with high value), especially comparing to Smokin' Guns which has an average 59% of CRFL for Type-1 Clones and and 66% for Type-3 Clones. Smokin' Guns achieves highest values for all the categories. Meanwhile, the average CRFL of the Python-based games are remaining in an acceptable level, i.e., under 40% for Type-3. The last noticeable observation is that the max. CRFL of Open Arena, Smokin' Guns, Unvanguished and World of Padman is 100% for Type-1 Clones. That means there are some files which are completely duplicated. Therefore, for software maintenance and refactoring purposes, we can increase the priority of those files if refactoring is going to happen.

To answer RQ4, we may conclude: *the average CRFL values for most of the games are low, except C-based games due to Type-1 cloning at file-level granularity. Also, the average CRFL of all games increases consistently for Gapped Clones (Type-3) which shows developers tend to customize the duplicated code fragments in most cases.*

E. RQ5 – Whether inter-game cloning happens?

In this section, we provide the result for CCH and CCP metrics related to the location of clones. As Kapser and Godfrey presented [3], the code clones could be classified into different categories based on the location information. We classify the code clones into intra-game code clones which means the code fragment forming a clone pair with another fragment within the same game, and inter-games code clones, which means the code fragment forming a clone pair with another fragment with different games. A higher CCH value denotes more intra-game code cloning occurrence in the games. Otherwise, code clones are to be considered as inter-game.

Table 2 summarizes (right most columns) the CCH and CCP values of all games for different types of clones. For Type-1, we notice almost all of Java and Python-based games are highly clone-cohesive, where CCH is close to 100%. That means the clones are almost intra-game clones. On the other hand, for the C-based games, the results are totally different. Except Hexen2, other C-based games are highly coupled, e.g., as high as 69% CCP for Unvanguished and World of Padman. Therefore, inter-game cloning is a common practice in open source C games development. Interestingly, although the C-based games are highly coupled in case of *exact clones* (Typr-1), the CCP remains stable (or decreases) comparing to Type-2 and 3. However, approaching to Type-3, most of Java and Python-based games become coupled, especially for Mediocre Chess (51% CCP for Type-3) and Sabacc (51% CCP for Type-3). In this case, CCP for most of such games increase up to ~20%.

To answer RQ5, we may conclude: *for C-based games, the CCP remains stable (relatively high) for all clone types. Therefore, there are lots of inter-games clones in the C games. For Java and Python-based games, there are more intra-game clones for Type-1 and 2, but there is low clone-cohesiveness when Type-3 is considered.*

TABLE II. CRFL, CCH, AND CCP METRICS (%) FOR OSS GAMES

Lang.	Genre	Game	Max CRFL			Min CRFL			Avg. CRFL			CCH			CCP		
			Research Question 4									*Research Question 5*					
			Type-1	*Type-2*	*Type-3*	*Type-1*	*Type-2*	*Type-3*	*Type-1*	*Type-2*	*Type-3*	*Type-1*	*Type-2*	*Type-3*	*Type-1*	*Type-2*	*Type-3*
C	First Person Shooter	Chocolate Doom	0.54	0.75	0.89	0.01	0.01	0.01	0.14	0.16	0.26	0.51	0.53	0.53	0.49	0.47	0.47
		Doom64 EX	0.56	0.75	0.80	0.01	0.01	0.01	0.17	0.16	0.25	0.51	0.57	0.56	0.49	0.44	0.44
		Hexen2	0.69	0.75	0.97	0.01	0.00	0.01	0.24	0.22	0.38	1.00	0.98	0.92	0.00	0.02	0.08
		Open Arena	1.00	1.00	1.00	0.01	0.01	0.02	0.53	0.52	0.63	0.38	0.38	0.35	0.62	0.62	0.65
		Smokin' Guns	1.00	1.00	1.00	0.01	0.01	0.03	0.59	0.59	0.66	0.32	0.31	0.29	0.68	0.69	0.71
		Unvanguished	1.00	1.00	1.00	0.00	0.00	0.00	0.34	0.29	0.37	0.31	0.37	0.42	0.69	0.63	0.58
		World of Padman	1.00	1.00	1.00	0.01	0.01	0.01	0.50	0.51	0.60	0.31	0.32	0.30	0.69	0.68	0.70
Java	Chess	jChecs	0.24	0.30	0.69	0.00	0.02	0.02	0.08	0.08	0.20	1.00	1.00	0.78	0.00	0.00	0.22
		jChess	0.13	0.24	0.72	0.02	0.01	0.01	0.05	0.09	0.16	1.00	0.91	0.69	0.00	0.09	0.31
		jose	0.58	0.58	0.68	0.00	0.00	0.01	0.10	0.11	0.15	1.00	0.98	0.85	0.00	0.02	0.15
		Mediocre Chess	0.00	0.63	0.68	0.00	0.04	0.06	0.00	0.25	0.24	0.00	0.90	0.49	0.00	0.10	0.51
	3D Engine	Ardor3d	0.42	0.54	0.91	0.01	0.01	0.01	0.11	0.14	0.25	0.97	0.96	0.77	0.03	0.04	0.23
		Env3d	0.09	0.01	0.59	0.00	0.22	0.03	0.05	0.11	0.21	1.00	0.98	0.65	0.00	0.02	0.35
		Jake2	0.79	0.79	0.83	0.00	0.01	0.00	0.22	0.20	0.30	1.00	1.00	0.93	0.00	0.00	0.07
		JMonkeyEngine	0.82	0.84	0.84	0.00	0.00	0.01	0.13	0.12	0.21	0.99	0.96	0.83	0.01	0.04	0.17
	Role Playing Game	Hale	0.41	0.41	0.62	0.01	0.01	0.01	0.08	0.07	0.17	1.00	0.93	0.78	0.00	0.07	0.22
		jClassicRPG	0.46	0.59	0.78	0.00	0.00	0.01	0.12	0.15	0.24	0.97	0.92	0.78	0.03	0.08	0.22
		OpenRPG	0.12	0.18	0.41	0.03	0.01	0.01	0.09	0.11	0.16	1.00	0.97	0.60	0.00	0.03	0.40
		Tower Zaldagor	0.40	0.40	0.98	0.01	0.01	0.01	0.09	0.13	0.24	1.00	0.99	0.79	0.00	0.01	0.21
Python	Card Game	Duo	0.39	0.41	0.69	0.01	0.01	0.13	0.19	0.13	0.34	1.00	0.90	0.61	0.00	0.10	0.39
		PySolFC	0.83	0.83	0.99	0.00	0.01	0.01	0.19	0.22	0.38	1.00	0.90	0.64	0.00	0.10	0.36
		Sabacc	0.00	0.04	0.55	0.00	0.04	0.02	0.00	0.04	0.20	0.00	1.00	0.49	0.00	0.00	0.51
		ScoPy	0.16	0.20	0.89	0.10	0.01	0.03	0.11	0.12	0.39	1.00	1.00	0.96	0.00	0.00	0.04
		Wizards Magic	0.33	0.33	0.77	0.03	0.01	0.01	0.23	0.13	0.26	1.00	0.98	0.67	0.00	0.02	0.33

F. RQ6 -Is there any duplicated code that could be reusable for future game development, such as extracting duplicated code to build new libraries for game developments?

In this subsection, we discuss how the cloned code, especially for inter-game clones, may be useful to the game development. In order to find the usable clone fragments required for this study, we manually classify 100 random Type-3 clones into two major categories: functionality similarity and framework usage.

First, functional similarities are those fragments from different games sharing similar functionalities. This kind of clones can be used as the source of creation of libraries, so they are the expected clones in the analysis. Second, some clones are caused by usage of the same framework or the existing open source libraries. We consider them as framework usage clones. The usage of framework is common reason for experiencing inter-program cloning.

By manual evaluation of the 100 Type-3 cloned functions, we identify 39 cloned functions belonging to the similar functionality category. Such detected clones can be used as the source to form new libraries for the future development of similar games. This observation supports the idea that the detected clones can be used in a positive way, since the presence of clones reduces game maintainability and increases development cost.

In summary, we answer RQ6 as follows: *the inter-game clones are made due to applying the same open source code or framework repeatedly. It is shown that automatic clone detection and management can be exploited successfully with high confidence to infer the foundations of new libraries.*

IV. THREATS TO VALIDITY

There are several threats to the validity in this study. First, there is no standard definition to similarity. However, in order to mitigate this threat we used one the state of the art clone detection tools, NiCad which detects Type-1, 2, and 3 clones with high precision and recall [2][38][39][40]. Second, the input data may also be a threat to the validity of this study. Therefore, we selected more than twenty games from different genres and languages to have a fair representative of OSS games. Third, the existence of false positive in the results may affect the accuracy of the analysis. In case of Type-3, we found a few false positives by manual inspection. The false positives are excluded from this study.

V. RELATED WORK

Several empirical studies related to cloning in open source systems are available in the literature. For example, Kapser and Godfrey [3] have provided a study with Apache httpd, the Linux file system and several other OSS to categorize the code cloning taxonomy. Similar studies have been done by Uchida et al. [33] and Rajapakse and Jarzabek [34], and Roy and Cordy [2]. Recently, Ishihara et al. [35] and Krinke et al. [36] reported comprehensive status on cloning in Java OSS and GENOME projects by highlighting potentials for clone detection and management in the domains of discourse.

While all of these studies (including this research) share similar approach, they report status of cloning in diverse domains and languages. Our study significantly differs from them since (1) we focus on game systems, (2) we exploit three different types of clones including both the exact (Type-1) and near-miss (Type-2 and Type-3) clones, and reported their status independently, and (3) we consider a variety of metrics.

VI. Conclusion and Future Work

In this paper, we have provided an in-depth study of the property of both exact and near-miss code clones in more than twenty open source C, Java and Python games from five categories, including First Person Shooter Games, 3D Game Engines, Chess Games, Role-Playing Games, and Card games. Using a selection of metrics from several different dimensions, we analyze the status of different types of code clones and duplication in those games. Specifically our study shows that (1) the characteristics of code clones in different categories are consistent to the building languages; (2) the average CRFL for most of the games is low, except some C-based games (exact clones), and (3) there are considerable number of inter-games clones in the C games and intra-game clones in Java-based and Python-based games. Our manual evaluation showed that they can be used as the seed to create new libraries related to game development. The result of our study is available online [37] for others to reuse the study. In summary, our research illustrates that cloning happens in intra-game cases as well as inter-games excessively. For example, there exists a large number of Type-1 clones in First Person Shooter games developed using C. This observation concretely shows the necessity of adapting clone management systems for game development. For future work, we plan to analyze code clones in distributed games.

References

[1] C.K. Roy, J.R. Cordy and R. Koschke. Comparison and Evaluation of Clone Detection Techniques and Tools: A Qualitative Approach. Science of Computer Programming, 74 (2009) 470-495, 2009.

[2] C. K. Roy and J. R. Cordy. Near-miss Function Clones in Open Source Software: An Empirical Study. Journal of Software Maintenance and Evolution: Research and Practice, 2(3): 165 – 189, 2010.

[3] C. Kapser and M. W. Godfrey. "Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software. Empirical Software Engineering, Vol. 13(6):645-692, 2008.

[4] E. Juergens, F. Deissenboeck, B. Hummel and S. Wagner. Do Code Clones Matter?. ICSE, 2009, 11 pp.

[5] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo. Comparison and Evaluation of Clone Detection Tools. IEEE Transactions on Software Engineering, 33(9):577-591, 2007.

[6] J.R. Cordy and C.K. Roy, 2011. The NiCad Clone Detector. Tool Demo Track, ICPC, 2011, pp. 219-220.

[7] M. Asaduzzaman. Visualization and Analysis of Software Clones. Master. Thesis, University of Saskatchewan, 109 pp., 2012.

[8] Chocolate Doom: http://www.chocolate-doom.org/ (Jun. 2012)

[9] Doom64 EX: http://doom64ex.wordpress.com/ (Nov. 2012)

[10] Hexen2: http://uhexen2.sourceforge.net/ (Oct. 2012)

[11] Open Arena: http://www.openarena.ws/ (Feb. 2012)

[12] Smokin' Guns: http://www.smokin-guns.org/ (Nov. 2012)

[13] Unvanquished: http://www.unvanquished.net/ (Nov. 2012)

[14] World of Padman: http://worldofpadman.net/ (Nov. 2012)

[15] jChecs: http://jchecs.free.fr/ (Oct. 2007)

[16] jChess: sourceforge.net/projects/jchesslibraryss/ (Jun. 2012)

[17] jose: http://jose-chess.sourceforge.net/ (Feb. 2011)

[18] Mediocre Chess: mediocrechess.sourceforge.net/ (Dec. 2009)

[19] Ardor3d: http://ardor3d.com/ (Nov. 2012)

[20] env3d: http://env3d.org/ (Nov. 2012)

[21] Jake2: http://www.bytonic.de/html/jake2.html (Jul. 2011)

[22] jMonkeyEngine: http://jmonkeyengine.com/ (Nov. 2012)

[23] Hale: http://sourceforge.net/p/hale/wiki/Home/ (Nov. 2012)

[24] jClassicRPG: http://javacrpg.sourceforge.net/ (May 2011)

[25] OpenRPG: www.rpgobjects.com/index.php?c=orpg (May 2010)

[26] Tower of Zaldagor: sourceforge.net/projects/toz (Oct. 2012)

[27] Duo: http://duo.berlios.de/ (Nov. 2012)

[28] PySolFC: http://pysolfc.sourceforge.net/ (Jun. 2011)

[29] Sabacc: http://sabacc.sourceforge.net/ (Oct. 2011)

[30] ScoPy: http://scopy.sourceforge.net/ (Jul. 2012)

[31] Wizards Magic: wizards-magic.sourceforge.net/ (Jul. 2011)

[32] R. Al-Ekram, C. Kapser and M. Godfrey, Cloning by Accident: An Empirical Study of Source Code Cloning Across Software Systems. ISESE, pp. 376-385, 2005.

[33] S. Uchida, A. Monden, N. Ohsugi and T. Kamiya. Software Analysis by Code Clones in Open Source Software. Journal of Computer Information Systems, XLV(3):1-11, 2005.

[34] D. C. Rajapakse and S. Jarzabek. An Investigation of Cloning in Web Applications. In WWW, pp. 924-925, 2005.

[35] T. Ishihara, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto. Inter-project functional clone detection toward building libraries - An empirical study on 13,000 projects. WCRE, 2012, pp. 387-391.

[36] J. Krinke, N. Gold, Y. Jia, D. Binkley. Cloning and copying between GNOME projects. MSR, 2010, pp. 98-101.

[37] Game Clones, http://www.cs.usask.ca/~croy/ GameClones.zip.

[38] J. Svajlenko, C.K. Roy, and J. Cordy. A Mutation Analysis Based Benchmarking Framework for Clone Detectors. International Workshop on Software Clones (IWSC), 2013.

[39] C.K. Roy and J.R. Cordy. A Mutation / Injection-based Automatic Framework for Evaluating Code Clone Detection Tools. International Workshop on Mutation (ICST), 2009.

[40] C.K. Roy and J.R. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. International Conference on Program Comprehension (ICPC), 2008.