Is Code Cloning in Games Really Different?

Farouq Al-omari
University of Saskatchewan, Canada
faa634@mail.usask.ca

Chanchal K. Roy
University of Saskatchewan, Canada
chanchal.roy@usask.ca

ABSTRACT

Since there are a tremendous number of similar functionalities related to images, 3D graphics, sounds, and script in games software, there is a common wisdom that there might be more cloned code in games compared to traditional software. Also, there might be more cloned code across games since many of these games share similar strategies and libraries. In this study, we attempt to investigate whether such statements are true by conducting a large empirical study using 32 games and 9 non-games software, written in three different programming languages C, Java, and C#, for the case of both exact and near-miss clones. Using a hybrid clone detection tool NiCad and a visualization tool VisCad, we examine and compare the cloning status in them and compare it to the non-games, and examine the cloned methods across game engines. The results show that code reuse in open source games is much different from that of other software systems. Specifically, in contrast to the common wisdom, there are fewer function clones in game open source comparing to non-game open source software systems. Similar to non-games open source, we observed that cloning status changes between different programming languages of the games. In addition, there are very fewer clones across games and mostly no clones (no code reuse) across different game engines. But clones exist heavily across recreated (cloned) games.

Keywords

Software clones, game clones, open source games

1. INTRODUCTION

Code cloning or code reusing is an inevitable practice by programmers [27]. Programmers usually copy and paste their code. In some cases programmers look through the Internet or related open source systems for a tested and ready to use source code. Programmers reuse the source code of open source systems not only for the low cost and ease of access, but also possibly for its quality. Large software systems contain 10-15% of duplicated code, which are considered as code clones [16].

Fowler and Beck [9] consider duplicated code as the #1 code bad smell. Furthermore, they describe a number of ways to remove clones to make programs easier to understand and maintain. Empirical studies [24, 18] report that while software clones are a principle reengineering technique and could be useful in many ways [16, 31], they are harmful to software and degrade software quality [8, 15, 4]. For example, if a code fragment contains some form of defects; all copies of that fragment should be located and fixed [27].

Open source systems are popular not only because of free access, but also their quality. Apache and Linux are good examples of open source systems both in terms of quality and popularity. Not only open community can leverage the benefits of open source systems, but also commercial software vendors exploit such invaluable resources [21]. Mockus [21] indicates that there is a large scale reuse of code between open source software that is represented in three forms: entire files, functions, and templates. Even core developers have the intention to reuse code across their projects at all stages of the development process. Also, they use tools to search and integrate the reused code fragments [11]. Therefore, the reused source code across open source systems in each domain need to be analyzed. However, an empirical study could not (possibly should not) cover all open source domains. Therefore, in this study we aim to conduct an empirical study for a growing open source domain, the games.

Usually, game software contains common functionalities related to 2D images, 3D models with animation, collision maps, sound, physics, and artificial intelligence. Also, most games are built upon application programming interface (API) such as OpenGL and libraries such as DirectX. Such commonality suggests that there may be more clones in games than other software systems.

Usually, game development has more challenges than development of other application-oriented noncritical software systems. Games need to be feature complete unlike other software that could be released with missing feature. Most games require a high level of graphic design and other challenges that pose programmer to clone some component from existing ones. Developers can copyright part of their project as maps or characters. However, it is not possible to protect the idea, software design, or game mechanics from cloning, in particular in the open source community. There is thus a conventional wisdom that there may be more cross games clones than other software systems.

SAC 2016, April 04-08, 2016, Pisa, Italy

http://dx.doi.org/10.1145/2851613.2851792 .

In this study we thus conduct an empirical study with 32 games of three different languages C, Java, and C# using one of the state of the art clone detectors, NiCad [25] and a visualization and analysis system, VisCad [2]. Furthermore, in order to compare the findings with non-game open source software, we use nine non-game software as of our previous study [29]. Our study aims to answer the following research questions:

RQ1: *What is the copy/pasted cloning (reusing either with direct copy or with modifications) status in open source games for different programming languages?*
RQ2: *How does the intensity of reusing of code fragments differ between games and non-game software systems?*
RQ3: *How often developers reuse functionalities from other related games?*

Our experimental results show that while there is indeed some evidence of copy/paste reuse in open source games, the extent of reuse is not that significant as compared to non-games and that there is little evidence of cross-game copy/paste reuse.

The remaining of the paper is organized as follows. Section 2 covers experiment setup. In Section 3 we demonstrate the selected clone related metrics. Section 4 provides experimental results that answer research questions. Section 5 discusses related works and finally Section 6 concludes the paper.

2. EXPERIMENTAL SETUP

Subject Systems: In this study, we analyzed 32 open source games for three different programming languages C, C#, and Java. Table 1 summarizes these systems. For each programming language we choose a number of open source games from different categories (e.g., shooting, strategy, and racing) and with different sizes. In addition, we compared the cloning/reusability status of the selected games open source to another group of a non-game open source. Therefore, we selected another group of non-game systems for this section of our study. Roy and Cordy [29] studied the clone status in these systems (i.e., non-games). Table 2 summarizes these subject systems.

Tool Settings: For the purpose of reusability/clone detection we used NiCad [25], a state-of-art clone detection tool. NiCad is a scalable and flexible clone detection tool with high recall and precision in detecting near-miss intentional clones [29]. Code clones can be categorized as Type-1, Type-2, Type-3, and Type-4 clones [27]. Type-1 clones are exact copies of each other, except for possible differences in whitespaces, comments and formatting. Type-2 clones are parameterized copies, where variable and function names, etc. have been renamed. Changes (e.g. additions, deletions and/or modifications of statements) in Type-2/Type-1 clone pair result in Type-3. Type-4 is syntactically dissimilar but semantically similar code fragments.

In this study, however, we did not aim at studying different types of clones. Rather, our aim was to study the extent of direct copy/paste with or without minor adaptations, which we call intentional clones [29]. We apply a standard pretty-printing using NiCad that removes the comments and makes the code consistent in terms of whitespaces and formatting. After that we used different similarity thresholds of NiCad (UPI thresholds). In particular, we used UPI thresholds of 0.0, 0.1, 0.2 and 0.3. When we use UPI threshold 0.0 on the pretty-printed code, we can find out the exact copy/pasted code. When UPI threshold 0.1 is used, we find almost similar code where there could be only 10% differences (i.e., 90% similar) between the cloned/reused fragments. Similarly, when UPI thresholds 0.2 and 0.3 have been used to determine the extent of similarity (20% and 30% differences respectively) and reuse of source code in the subject systems. Again, the benefits of using pretty-printing and these thresholds is that they help us find the intentional copy/paste/reuse code fragments instead of just similar fragments.

In order to analyze the detected clones we used VisCad [2], a clone visualization tool that supports a number of clone detectors. In addition to visualization, VisCad offers a number of clone related metrics that facilitate analysis of clones.

Table 1: Selected game subject systems in the study

Lang	Subject system	NOF	LOC	NOM
Java	Freecol	689	193k	8472
	Robocode	653	94k	5502
	AndEngine	596	66k	4621
	Greenfoot	345	60k	2907
	Terasology	614	86k	5754
	Env3D	75	8k	741
	Jake2	258	85k	4160
C#	Mono Game	791	166k	3587
	Unity Steer	51	515k	115
	OpenRA	630	643k	2121
	Samurai	82	568k	228
	Sleep Walker	528	616k	2010
	NetGore	603	66k	2016
	Axiom Engine	421	121k	4273
C	Freedroid	128	77k	2414
	Berlios	188	162k	2601
	Quake 3	428	347k	7798
	Open Arena	171	147k	3232
	Egoboo	179	194k	1636
	Nethack	254	222k	2100
	Chocolate Doom	174	103k	1792
	FreeCiv	459	285k	11330
	SGE	201	44k	342
	Hexan	56	70k	1224
	Wolf3D	28	15k	558
	Chocolate Doom	159	86k	1771
	Quake 2	181	149k	4472
	Enemy Territory	456	473k	9096
	QW	112	78k	1780
	WinQuake	127	88k	2082
	Quake 2	181	149k	4477
	Quake 3	428	348k	7798

Table 2: Non-games subject systems

Lang	Subject system	NOF	LOC	NOM
Java	Eclipse- jdt core	1202	455k	18257
	CIRC	65	127k	765
	JHotDraw	469	189k	2886
C#	Mono c# compiler	54	880k	1993
	Castle	2407	270k	9236
	Nant-0.68	429	104k	2335
C	Apache http	252	200k	3878
	Postgresql	1155	1018k	15399
	Cook	296	73987	1335

3. CLONE METRICS

Total Cloned Method (TCM): It is common to measure clone density in a system by the number of clone pairs and clone classes. However, in this metric we are using the total number of cloned methods in the system to represent clone density. A method is considered cloned method if it forms at least a clone pair with another method in the system. We also used Total Cloned Method Percentage (TCMp), the percent of cloned methods in the system.

Total Cloned Line of Code (TCLOC): Since some cloned methods are small and others are large in size we considered the cloned method size (Lines of Code, LOC) as a metric for our study. A Total Cloned Line of Code (TCLOC) metric is used to determine the total cloned line of code in the system. Another inherited metric also used *TCLOCp* that represents the percentage of cloned line of code in the system.

File Associated with clones (FAWC): The previous metrics measure the number and percentage of cloned methods and its size. However, it is still important to know the portion of files that hold these clones and how/where these clones are distributed in the system. Therefore, FAWC reports the number of files that contain at least a cloned method. Also, *FAWCp* is defined as the percentage of files associated with clones in the system.

Clone Class Radius (CCR): Previous metrics show the density of clones in the subject system. We still need to measure the distribution of these clones in the system. Thus by clone class radius we mean the average distance between cloned files in clone classes and their lowest common ancestor directory in the file system. The larger CCR the more scattered cloned file class in the system; therefore, the more maintenance effort. Also, we refer to Clone Class Size as CCS which is the number of cloned fragment in the class. Other code metrics that we used: (1) LOC: Lines of Code, (2) NOF: Number of Files, (3) NOM: Number of Methods, (4) NOCC: Number of Clone Classes, (5) NOCP: Number of Clone Pairs.

4. EXPERIMENT RESULTS

In this section we discuss the experiment results of the study. Each subsection of the experiment is designed to answer a research question. In discussion we provide the overall findings and describe the statistical measures. The following subsections describe in details our results and findings.

4.1 Cloning Status in Game Open Source

In this subsection, we answer ***RQ1*** by reporting our findings regarding clone status in open source games developed using three different programming languages. We carried out an extensive analysis on 32 systems using the selected metrics (Section 3) to have deeper insight into the clone status in open source games.

4.1.1 Overall cloning density

In this subsection, we provide the overall cloning level for different games open source systems in different programming languages. Although, there is debate about the effect of clones in software, clones are likely harmful to the system and they increase the effort of maintenance and refactoring [22, 19]. Figure 1 shows the percentage of cloned functions for each programming language systems in total. First, the figure shows that C# games have the highest percentage of cloned method and Java open source games have the lowest cloned method while C open source games fall in between. As the value of NiCad UPI increases the cloned method percent increases for all programming languages. However, cloned method in C systems has a higher increasing rate than other Java and C# and its cloned method percent becomes larger than C# for UPI=0.3, which means it has more near-miss clones.

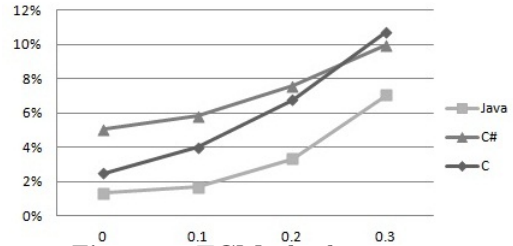

Figure 1: TCMp by language.

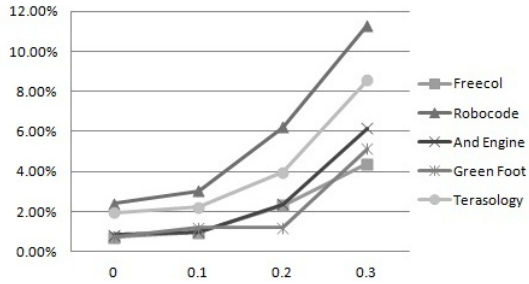

Figure 2: TCMp for Java games

There is a clear difference of cloning level between languages. However, we still want to see if there are individual differences between games in the same language. In order to interpret these differences, we analyzed each individual system clones as shown in Figures 2-4 (the TCMp values). An expected observation is that the existence of variation in cloning level for the same language games. But, this variation is still less than the variation between languages.

Figures 5, 6, and 7 show the TCLOCp for each system in the target programming languages. In these figures the general trend is the same as in the corresponding TCMp figures. These figures emphasize the same facts that have been concluded from TCMp metric. There is a variation in cloning levels across programming language games. In addition, there is a variation in cloning level for the same language games. But, this variation is still less than the variation between languages. Both TCMp and TCLOCp have the same order of games according to their cloning level. Figure 8 shows the TCLOCp by language and Table 3 shows the detailed value of TCM, TCLOCp, and NOCC for each game.

4.1.2 Clone Associated Files

As a quality measure, the smaller the FAWCp the better system quality is. Figure 9 shows the average FAWCp by language. Identical clones are more spread in C# games followed by C games and least spread in Java games. By looking at TCMp in Figure 1, we have the same language order according to cloning level. The interesting thing in comparing these figures (Figure 1 and Figure 9), we find that the relationship between TCMp and FAWCp is not followed as we moved toward near-miss clones. Near-miss Java clones spread out more rapidly in game files and exceed the FAWCp in C#. In other words, the near-miss clones in C# games reside in the same files that contain identical clones. Even C# games have the highest percent of FAWC for identical clones that is a good indication of the game quality. To conclude, C games are considered to need more maintaining efforts than other languages since it has more cloned copies through its source files.

Table 3: NOCC, TCM, and TCLOC for all games

		Freecol			robocode			Andengine			greenfoot			Terasology		
Java	UPI	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp
	0.3	141	372	2.65	160	621	9.1	98	284	5.92	64	149	3.9	149	492	6.96
	0.2	81	196	1.48	122	342	6.01	41	109	2.31	35	35	0.97	86	228	3.07
	0.1	37	80	0.67	64	167	3.44	20	46	0.99	17	35	0.97	57	127	1.66
	0	28	60	0.38	54	132	2.75	16	38	0.61	11	22	0.52	50	112	1.43
		Mono Game			Unity Steer			OpenRA			Samurai			Sleep Walker		
C#	UPI	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp
	0.3	210	553	8.6	2	4	2.74	26	62	1.44	7	18	3.89	66	165	7.1
	0.2	183	461	7.12	1	2	2.37	15	37	0.8	7	15	3.11	41	96	4.5
	0.1	146	372	5.76	1	2	2.37	8	22	0.49	5	11	2.37	26	62	2.87
	0	136	341	5.17	1	2	2.37	6	18	0.33	5	8	1.25	15	37	1
		Freedriod			Berlios			Quake 3			Openarena			Egoboo		
C	UPI	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp
	0.3	26	61	1.22	83	201	4.17	466	0.13	12.43	178	457	10.15	97	234	4.78
	0.2	4	9	0.15	44	93	2.12	360	0.1	9.96	130	294	7.68	53	114	3.23
	0.1	1	3	0.05	19	38	0.82	239	0.07	7.17	82	170	5.65	30	61	1.88
	0	1	2	0.03	8	16	0.17	147	0.04	2.11	55	110	2.5	17	34	0.51
		NetHack			Chocolate Doom											
	UPI	NOCC	TCM	TCLOCp	NOCC	TCM	TCLOCp									
	0.3	64	149	2.36	41	94	2.95									
	0.2	46	101	1.82	18	40	1.57									
	0.1	31	67	1.33	7	14	0.69									
	0	27	58	1.19	3	6	0.16									

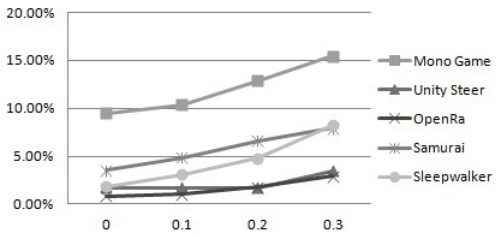

Figure 3: TCMp for C# games

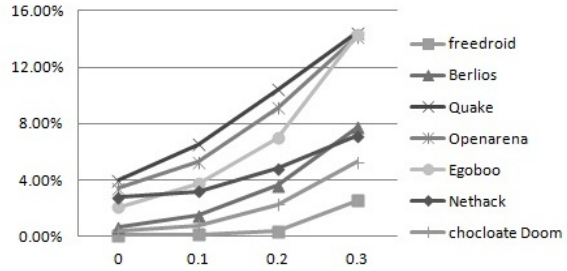

Figure 4: TCMp for C games

4.1.3 Profile of Cloning Localization

As mentioned early, the better the system quality the less cloning level is and the less number of files contain clones. Furthermore, in this section, we discuss a metric for clone localization. As a measure of quality, the more localized the clones are the better quality of the system is, leading to less maintenance efforts and refactoring. We are using Clone Class Radius (CCR) as a measure of cloning localization. CCR is calculated for each clone classes in the system as described in Section 3. The average for all clone class is considered the average CCR for the system.

Figure 10 compares CCR by programming language through different UPI thresholds. Basically, an important factor that affects the value of the CCR is the nature and the size of system hierarchal structure. In our experiment we had chosen different size systems for each programming language to avoid the effect of system size. The CCR value for small systems that have only one folder would be equal to zero. Two facts are to discuss from Figure 10. First, C games have smaller CCR. One important factor that has an affect is that C games are smaller than other Java and C# games. The

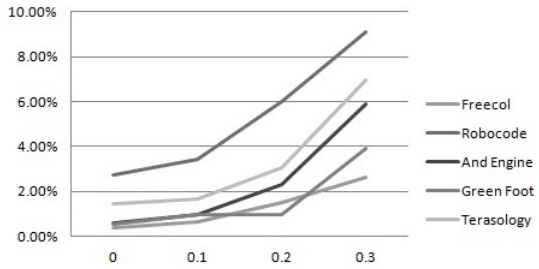

Figure 5: TCLOCp for Java games

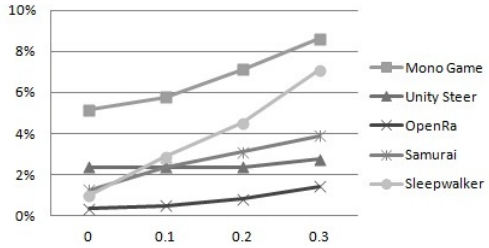

Figure 6: TCLOCp for C# games

second and interesting fact is the average class's radius for identical clones that is larger than near-miss clones, which means, near-miss clones reside on the same locations as of identical clones. We also analyzed the the CCR for each game individually and we find there is a high variation of CCR for games of the same programming language. The main facts to conclude from this subsection are: (1) clones in C games are more localized than clones in C# and Java games, and (2) CCS has an inverse relationship with CCR.

Summary of answer to RQ1: *There are clear differences in cloning density among the open source games with respect to programming languages. Our result reveals that games developed in C have more clones than games developed in C# and Java. However, this variation is less among games of the same programming languages. In addition, games developed in C have more percentage of files associated with clones and those files are more localized as compared to the games in other languages we analyzed. This suggests that game developers should be aware of the clones and their impacts on the systems in general and the developers using C language for games may need additional cautions.*

4.2 Game vs. Non-Game

To answer ***RQ2***, we compared cloning status in games

Table 4: Games vs. non-games clone status

		System	NOCC	NOM	TCM	TCMp	LOC	TCLOC	TCLOCp	NOF	FAWC	FAWCp	AvgCCR	AvgCCS
C	Non-Games	Apache http	138	3878	341	8.79	200K	11666	5.82	252	73	28.97	0.54	2.47
		Postgresql	3255	15399	2204	14.31	1018K	81939	8.04	1155	398	34.46	0.37	2.91
		Cook	74	1335	198	14.83	73K	5229	7.07	296	110	37.16	0.18	2.68
		All	3467	20612	2743	13.31	1293K	98834	7.64	1703	581	34.12	0.37	2.89
	Games	Freedroid	26	2414	61	2.53	77K	951	1.22	128	25	19.53	0.04	2.35
		Berlios	83	2601	201	7.73	162K	6790	4.17	188	54	28.72	0.24	2.42
		Quake	466	7798	1124	14.41	348K	43271	12.43	428	209	48.83	0.68	2.41
		Openarena	178	3232	457	14.14	147K	15014	10.15	171	75	43.86	0.32	2.55
		egoboo	97	1636	234	14.3	196K	9413	4.78	179	37	20.67	0	2.41
		Nethack	64	2100	149	7.1	222K	5248	2.36	254	42	16.54	0.23	2.33
		Choclate Doom	41	1792	94	5.25	103K	3062	2.95	174	38	21.84	0.22	2.29
		All	955	21573	2320	10.75	1259K	83749	6.65	1522	480	31.54	0.44	2.43
Java	Non-Games	Eclipse jdt Core	709	18257	2283	12.5	455k	49773	10.94	1202	479	39.85	1.1	3.22
		CIRC	13	765	33	4.31	12k	361	2.82	65	15	23.08	0.62	2.54
		JHotDraw	56	2886	173	5.99	189k	1980	1.05	469	70	14.93	0.45	3.09
		All	778	21908	2489	11.36	657k	52114	7.93	1736	564	32.49	1.04	3.2
	Games	Freecol	141	8472	372	4.39	193k	5113	2.65	689	170	24.67	0.18	2.64
		Robocode	160	5502	621	11.29	94k	8583	9.1	653	201	30.78	2.68	3.88
		AndEngine	98	4621	284	6.15	66k	3926	5.92	596	102	17.11	0.26	2.9
		GreenFoot	64	2907	149	5.13	60k	2346	3.9	345	64	18.55	0.2	2.33
		Terasology	149	5754	492	8.55	86k	6034	6.96	614	93	18.55	0.2	2.33
		All	612	27256	1918	7.04	500k	26002	5.19	2897	630	21.75	0.85	2.9
C#	Non-Games	mcs	20	1993	52	2.61	88k	1065	1.21	54	14	25.93	0	2.6
		Castle	329	9236	908	9.83	270k	18496	6.84	2407	302	12.55	0.51	2.76
		nant	84	2335	216	9.25	104k	4380	4.18	429	88	20.51	0.15	2.57
		All	433	13564	1176	8.67	463k	23941	5.17	2890	404	13.98	0.42	2.72
	Games	monogame	210	3587	553	15.42	166k	14301	8.6	791	228	28.82	1.33	2.63
		Unity Steer	2	115	4	3.48	5k	141	2.74	51	4	7.84	0	2
		OpenRA	26	2121	62	2.92	64k	926	1.44	630	49	7.78	0.73	2.38
		Samurai	5	228	18	7.89	5k	221	3.89	82	7	8.54	0	3.6
		Sleepwalker	66	2010	165	8.21	61k	4379	7.1	528	84	15.91	0.14	2.5
		All	309	8061	802	9.95	303k	19968	6.59	2082	372	17.87	1	2.6

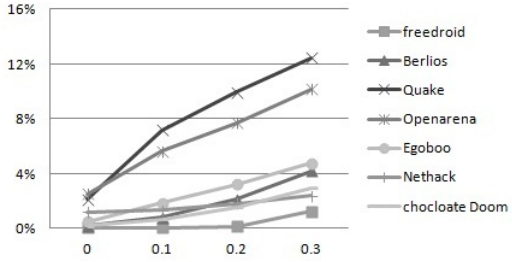

Figure 7: TCLOCp for C games

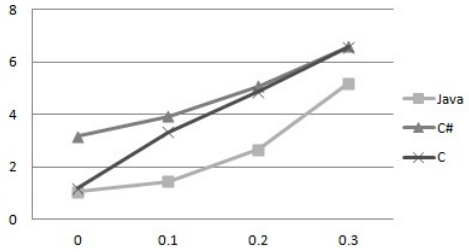

Figure 8: TCLOCp by language

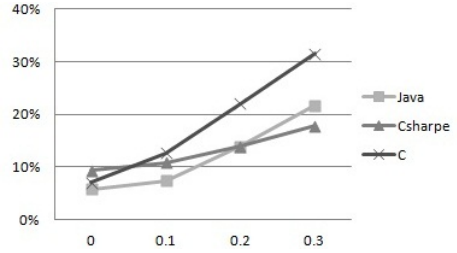

Figure 9: FAWCp by language

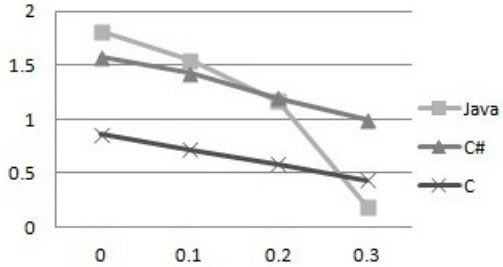

Figure 10: Average CCR by language.

with that of non-game systems. We chose three popular open sources non-games for each programming language. Originally, these systems were used by Roy and Cordy's [29] empirical study on clones. Table 2 shows these systems. We applied the same metrics as the previous section. Table 4 shows the metrics for these systems and the corresponding metrics for open source games. The most interesting observation is that in general the cloning level for all open source non-games is higher than all open sources games. Also, it shows the CCS is larger in non-game open source and the CCR is smaller that shows games have better quality in term of cloning level and distribution in file system.

Summary of answer to RQ2: *First, results for the three cloning level metrics used (TCMp, TCLOCp and FAWCp) indicate that there are more clones in open source non-games systems as compared to open source games. Second, clones in non-games are more localized than in games open source. The average CCR for all non-games open source is 0.37 while it is 0.44 in games. Finally, there is no clear deviance that games have less files associate with clones.*

4.3 Clones across Game Engines

In this section we answer ***RQ3*** by analyzing cloning status across game engines to show the amount of shared code in game open source and whether game developers are used to copy code from another game. The experiment in this

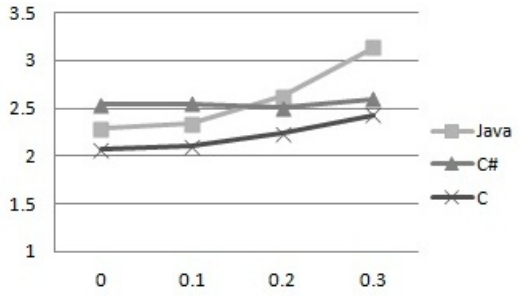

Figure 11: Average CCS by language

Table 5: clone status across unrelated games

	Game	NOCC	NOCP	TCM	TCMp
C	FreeCiv	342	1175	905	7.99
	SGE	9	9	18	5.26
	Both	351	1184	928	7.95
Java	Env3D	16	20	34	4.59
	Jake2	246	717	660	15.87
	Both	259	737	694	14.16
C#	NetGore	91	326	235	11.66
	Axiom Engine	163	1408	506	11.84
	Both	254	1734	741	11.78

section is divided into three parts according to engine selection. We studied clone status between unrelated engines, engines of the same game genre, and recreated games.

4.3.1 Clones across unrelated game engines

The purpose of this experiment is to study the clone status across unrelated -not belonging to the same category games. Therefore, we chose two unrelated game engines for each programming language. For Java, we had chosen Env3D and Jake2 Game engines. We selected FreeCiv2.3.3 and SGE2D for our C language case study. Finally, we chose NetGore and Axiom game engines for C#. Our experiment started by an analysis of clones for each game independently. Then we combine each two games of the same programming language as one system and detect and analyze its clones. Thus, the number of cross game clones can be defined as clones for combined games A and B -(clones of game A + clones of game B).

Table 5 summarizes the results for this experiment. The results indicate there is no cross game engine clones for all cases that we had studied. For example, in C game engines there are 1184 clone pairs where 1175 from the first engine, 9 from the second engine and no clone pairs exist between the two engines.

4.3.2 Same genre game engines clones

We studied clone status between unrelated game engines above. The next question is: what is the cloning status between game engines that belong to the same category. To answer this question, we have to find a number of open source game engines that belong to the same category. Id Software[1] developed a number of game series that all classified first person shooter. Started from Vintage series through Wolfenstein, Doom, Quake, and Enemy Territory series and ended by mobile series. Each of the mentioned series has a number of games. Thus, we select one game from each series for this experiment. We select Hexan, Wolf3s, Chocolate Doom, Quake 2, and Enemy Territory.

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

Table 6: Clone status for same category game engines

Game engine	NOCC	NOCP	NOCP cross games	TCM	TCMp	TCL OCp
Hexan	35	78	0	95	7.77	3.87
Wolf3D	22	36		50	8.97	9.37
Both	57	114		145	8.1369	4.84
Wolf3D	22	36		50	8.97	9.37
Chocolate Doom	40	60		89	5.03	3.44
Both	62	96	0	139	5.9682	4.33
Chocolate Doom	40	60		89	5.03	3.44
Quake 2	601	1027		1337	29.89	32.05
Both	641	1087	0	1426	22.841	21.59
Quake 2	601	1027		1337	29.89	32.05
Enemy Territory	465	1000		1116	12.27	8.49
Both	1072	2144	117	2511	18.5067	14.36

Table 7: Clone status across recreated games

System	NOCC	NOCP	NOCP cross games	TCM	TCMP	TCL OCp
QW	148	257		374	21.01	15.01
WinQuake	177	268		399	19.16	14.14
Both	837	1792	1267	2031	52.59	47.86
WinQuake	177	268		399	19.16	14.14
Quake 2	596	1039		1328	29.66	32.27
Both	879	1630	323	2028	30.92	47.86
Quake 2	596	1039		1328	29.66	32.27
Quake 3	466	1018		1124	14.41	12.43
Both	1079	2288	229	2028	30.92	19.11

We did a similar experiment to the previous section. We first analyze clones of each game independently. Then we combine each two games that belong to a consecutive series as a single dataset and detect and analyze its clones. Table 6 shows the clone status in each individual game and the results of combined games.

Tracing Table 6 not only shows how cloning software games engine occurs across Id Software games but also it shows the program evolution in them. It shows that the next engine is usually larger in term of LOC and number of methods except for Enemy Territory engine. Furthermore, the cloning level is more in the successor engines in term of TCMp and TCLOCp except for Enemy Territory game. Also, CCR increased in successor games. We are interested in the number of cross-engine clones. An interesting finding is that the first three game engine pairs show there are no cross-engine clones in spite of the fact that they are of the same category (first person shooter games) and of the same vendor. However, there are a limited number of clone pairs, 117 clone pairs, which is 5.5% of the total number of clone pairs, shared between Quake2 and Enemy Territory games. This implies there are no or a limited number of clones exist between games of the same family. This is an interesting and valuable observation since they are holding a low cloning level across systems comparing to cloning level in non-game open source [8, 9, 21, 28, 29].

4.3.3 Clones across recreated games

In previous sections, we have shown the clone status across games that are not related and games that belong to the same genre. Open Source community and game market place are full of cloned or recreated games. Do these games

share a common code clones base? In this section we are studying clone status in recreated games. In general, Id Software games are large series systems. For example, Quake series have 15 main versions in the main series starting from the main quake game that appears in 1996 and ending by quake Xbox game. However, quake family contains large number of games that are derived from the main series. In this section we study clone status in Quake series from id Software. Two facts could be conducted from Table 7. The first fact is that the four selected games have a similar clone status, TCMp, TCLOCp, FAWCp and CCS. The second fact is the existence of a high copy/paste cloning between games. For example, TCMP for QW is 21.01% and for WinQuake is 19.16%. But for two games together, it is 52.59%. It is clear that TCMp and TCLOCp increase due to cross games clones. The forth column in the table shows the number of cross-games clone pairs.

Our study that shows there are lots of clones between series games. It seems reasonable because the games in the series are evolved (cloned) from each other. In such cases where there are common functionalities across game series or recreated games, it would be beneficial to encapsulate those functionalities into a library. This library would be useful for other game series or other game developers.

Summary of answer to RQ3: *There is a clear evidence that there is less practice of copying source code across games open source. However, the cloning density is very high between games in the same series.*

5. RELATED WORK

Empirical studies of clones in open source source has been a popular topic and there have been a great many studies [26]. For example, when a new tool is published, it comes with an empirical study for validating the corresponding tool [27, 26]. Empirical studies of clones have also been conducted as part of tool comparison experiments such as the Bellon's benchmark [5]. Kapser and Godfrey have conducted several studies on clones, e.g., for deriving a taxonomy of clones [14] and then studying the harmfulness / usefulness of clones [16]. There are also a number of studies that study cloning and their evolution for deriving different maintenance implications of clones [26].

AL-Ekram et al. [1] conducted an empirical study of source code cloning across software systems. They analyzed clones between software systems in two domains: Text Editors and Window Managers. They used CCFinder as clone detector for Type 1 and 2 clones. Their results indicate that, there is not as many clones across systems and most of these clones are accidental rather than copy and paste clones. On the other hand, Ishihara et al. [13] did an empirical study for inter-project method clones for Java projects. Their method is based on hashing the generated value from AST (Abstract Syntax Trees). Although they did not consider domain-related analysis of the methods and they investigated only the top 100 shared method sets, their judgment indicates a 56 cloned method set are suitable as a library candidate.

There have been also a few studies that partly used open source games. For example, Gabel and Su [10] used four games out of 30 systems to study uniqueness of source code. Another clone related study that uses games presented by Schulze and Apel [30]. They conducted an empirical study to analyze the relationship between feature-oriented programming and clones. Furthermore, some studies [19, 10, 17, 20, 3, 12] used the source code of Freecol game engine as part of their target systems.

Munro et al. [23] analyzed the structure of Quake game engine series. Similar to our work, part of their study was detecting how much shared code was between Quake 1 and other Quake games. For this purpose, they use Simian to detect clones between Quake1 and other Quake games. They found that the amount of shared code decreases between Quake 1 and consecutive Quake game engines. But, the purpose of their study is to analyze the structure and evolution of Quake games.

Ciancarini and Favini [7] detect chess games that are cloned or derived from other chess games. Their approach was based on performing tournaments between tested games and candidate games are gone through code clone analysis then the final decision is made by expert referees. Their work shows the importance of code clone analysis in detection of cloned or plagiarized games.

Chen et al. [6] studied a number of open source games and is related to ours to some extent. However, again they focused on studying different types of clones such as the presence of Type 1, Type 2 and Type 3 clones in games. Our focus is on the other hand, to study the intentional copy/paste and reuse of program code between games and game engines. For detecting, Type 2 and 3 clones, one needs to normalize the source code before comparison, and thus the results may not reflect the reuse of code fragments among the subject software systems. As noted earlier, we use pretty-printing and then different similarity thresholds for understanding the extent of copy/paste and reuse in them. We also provide an extensive analysis with different game engines and attempted to find out interesting insights. Furthermore, we provide an extensive comparison with open source non-games systems for studying varying nature and extent of reusability between them.

The most closely related work to our empirical study is the work of Roy and Cordy [29]. They conducted an empirical study on near-miss clones reusability in open source software and that we use kind of similar metrics. However, in their study there was no domain-specific clone analysis and they did not use any game open source, whereas our focus is on games and game engines and then provide the comparative analysis with non-game open source software.

6. THREATS TO VALIDITY

One of the threats of our study would be the clone detector used for studying the copy/paste and reuse. We have used one of the state of the art clone detectors, NiCad which has been shown to give results with high precision and recall [28] not only for different types of clones but also for studying the copy/paste reusability for different similarity levels [29]. Another threat would be the limited number of samples. However, to the best of our knowledge this is the largest study on cloning reusability in games and that we further compared with non-games software. While we used NiCad with its pretty-printing and similarity thresholds, there is no guarantee that these returned clones are in fact the right ones for software maintenance activities such as refactoring. However, our target was just to find out the extent of reusability and not to find out clones that could be used for any maintenance activities.

7. CONCLUSION

With the fast growth of open source systems in different software domains, including game domains, there has been a need to analyze and compare the quality, maintainability, cloning status, and refactoring for each domain. In this paper, we perform an empirical study to analyze the cloning status in and across open source games and compared the results with other open source non-games. We analyzed clones in more than 32 games open source of three different languages.

The findings of our study shows that there is a considerable variation of cloning status (density, association with file, and localization) between games open source of different programming languages. On the other hand, the variation of cloning level between open source games of the same language is negligible. When comparing game open source to non-game open source, games have fewer clones than non-game open source and their clones are more localized. Finally, game open source domain tends to have less cross-game clones comparing to other open source systems. However, a lot of clones exist across recreated games. In such cases, code clone detection tool could be used to identify recreated (cloned) games.

8. REFERENCES

[1] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by accident: an empirical study of source code cloning across software systems. In *ESEM, 2005.*, pages 363–372, 2005.

[2] M. Asaduzzaman, C. K. Roy, and K. A. Schneider. VisCad: Flexible Code Clone Analysis Support For NiCad. In *IWSC*, page 77–78, 2011.

[3] H. A. Basit, U. Ali, and S. Jarzabek. Viewing simple clones from structural clones' perspective. In *IWSC*, page 1–6, 2011.

[4] S. Bazrafshan. No clones, no trouble? In *IWSC*, pages 37–38, 2013.

[5] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *TSE*, 33(9):577–591, 2007.

[6] Y. Chen, I. Keivanloo, and C. K. Roy. Near-miss software clones in open source games: An empirical study. In *CCECE*, pages 1–7, 2014.

[7] P. Ciancarini and G. P. Favini. Plagiarism detection in game-playing software. In *FDG*, page 264–271, 2009.

[8] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *ICSM*, pages 109–118, 1999.

[9] M. Fowler. *Refactoring: Improving the design of existing code*. Pearson Education India, 1999.

[10] M. Gabel and Z. Su. A study of the uniqueness of source code. In *FSE*, pages 147–156, 2010.

[11] S. Haefliger, G. von Krogh, and S. Spaeth. Code Reuse in Open Source Software. *Management Science*, 54(1):180–193, 2008.

[12] K. Hotta, Y. Sasaki, Y. Sano, Y. Higo, and S. Kusumoto. An Empirical Study on the Impact of Duplicate Code. *Advances in Software Engineering*, 2012:1–22, 2012.

[13] 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. In *WCRE*, pages 387–391, 2012.

[14] C. Kapser and M. W. Godfrey. Toward a taxonomy of clones in source code: A case study. *ELISA*, 2003.

[15] C. J. Kapser and M. W. Godfrey. Supporting the analysis of clones in software systems: Research articles. *J. Softw. Maint. Evol.*, 18(2):61–82, 2006.

[16] C. J. Kapser and M. W. Godfrey. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13(6):645–692, 2008.

[17] I. Keivanloo, C. K. Roy, and J. Rilling. Java bytecode clone detection via relaxation on code fingerprint and semantic web reasoning. In *IWSC*, pages 36–42, 2012.

[18] M. Kim, V. Sazawal, and D. Notkin. An empirical study of code clone genealogies. *ACM SIGSOFT Software Engineering Notes*, 30(5):187, 2005.

[19] A. Lozano and M. Wermelinger. Assessing the effect of clones on changeability. In *ICSME*, pages 227–236, 2008.

[20] A. Lozano and M. Wermelinger. Tracking clones' imprint. In *IWSC*, pages 65–72, 2010.

[21] A. Mockus. Large-Scale Code Reuse in Open Source Software. In *FLOSS'07: ICSE Workshops 2007*, pages 7–7, 2007.

[22] M. Mondal, M. S. Rahman, R. K. Saha, C. K. Roy, J. Krinke, and K. A. Schneider. An Empirical Study of the Impacts of Clones in Software Maintenance. In *IWPC*, pages 242–245, 2011.

[23] J. Munro, C. Boldyreff, and A. Capiluppi. Architectural studies of games engines –The quake series. In *ICE-GIC*, pages 246–255, 2009.

[24] M. S. Rahman and C. K. Roy. A Change-Type Based Empirical Study on the Stability of Cloned Code. In *SCAM*, pages 31–40, 2014.

[25] C. Roy and J. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *IWPC*, pages 172–181, 2008.

[26] C. Roy, M. Zibran, and R. Koschke. The vision of software clone management: Past, present, and future (keynote paper). In *CSMR-WCRE*, pages 18–33, 2014.

[27] C. K. Roy and J. R. Cordy. A Survey on Software Clone Detection Research. *Queen's School of Computing*, 541:115, 2007.

[28] C. K. Roy and J. R. Cordy. A mutation/injection-based automatic framework for evaluating code clone detection tools. In *ICSTW*, pages 157–166, 2009.

[29] C. K. Roy and J. R. Cordy. Near-miss function clones in open source software: an empirical study. *Journal of Software: Evolution and Process*, 22(3):165–189, 2010.

[30] S. Schulze, S. Apel, and C. Kästner. Code clones in feature-oriented software product lines. *ACM SIGPLAN Notices*, 46(2):103, 2011.

[31] M. Toomim, A. Begel, and S. Graham. Managing duplicated code with linked editing. In *VL/HCC*, pages 173–180, 2004.