A Study of Code Clone on Open Source VR Software

Wenjie Huang
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
huangwenjie241@mails.ucas.ac.cn

Jinfu Chen
Wuhan University
Wuhan, China
jinfuchen@whu.edu.cn

Huashan Chen*
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
chenhuashan@iie.ac.cn

Zhenyu Qi
University of Arizona
Tucson, USA
qzydustin@arizona.edu

Xiaojia Yang
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
yangxiaojia@iie.ac.cn

Kebin Peng
East Carolina University
Greenville, USA
pengk24@ecu.edu

Sen He
University of Arizona
Tucson, USA
senhe@arizona.edu

Abstract

Virtual reality (VR) offers unprecedented user experiences across diverse areas. However, few studies have focused on code clone in VR software, though it is a significant factor in the emergence and spread of security vulnerabilities in open-source software. In this work, we conduct the first extensive quantitative empirical analysis of code clone in open-source VR software. We create a dataset of 83 open-source VR projects sourced from GitHub and propose to use multiple metrics to facilitate quantitative analysis. Using the NiCad code clone detection tool, we provide a thorough analysis of code clone in our dataset to tackle five meticulously crafted research questions. Our analysis of the experimental results reveals various insights, enhancing our understanding of VR software and paving the way for future research endeavors.

CCS Concepts

• **Security and privacy** → *Software and application security*.

Keywords

Virtual Reality (VR), VR security, code clone

ACM Reference Format:
Wenjie Huang, Jinfu Chen, Huashan Chen, Zhenyu Qi, Xiaojia Yang, Kebin Peng, and Sen He. 2024. A Study of Code Clone on Open Source VR Software. In *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3691621.3694957

*Correspondence: chenhuashan@iie.ac.cn

ASEW '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1249-4/24/10
https://doi.org/10.1145/3691621.3694957

1 Introduction

Virtual reality (VR) techniques have provided revolutionary user experience in various application scenarios (e.g., training, education, advertisement, gaming, healthcare, manufacturing)[1, 2]. According to a recent report from Mordor Intelligence[3], the virtual reality market size is estimated at 22.81 billion dollars in 2024, and is expected to grow by 42.05% per year. While the majority of the VR market is still on hardware, the VR software market is experiencing significant growth and development, with thousands of applications being developed and uploaded to app distribution platforms(e.g., Google Play, Apple Store, Oculus), having been downloaded by about 200 million users worldwide[4].

While VR has advanced rapidly, it's important to recognize that its core concept and underlying technologies aren't entirely new. Most VR applications are running on top of off-the-shelf mobile operating systems (e.g., Android, Apple visionOS, Sony Orbis OS). As a result, VR applications share generic software security vulnerabilities with conventional mobile applications[5]. In addition, VR applications possess unique immersive characteristics, causing various VR-specific security vulnerabilities[6]. Unfortunately, the rapid expansion of VR applications has outpaced the research dedicated to their security. This significant gap poses a threat to user trust and satisfaction, ultimately jeopardizing the long-term success and sustainability of the VR industry. Specifically, one area that remains under-explored is code clone in VR software.

The reuse of identical or similar code fragments, known as code clone[7], is a common practice in software development, which is generally considered to be an important cause of introducing and propagating open source software security vulnerability[8]. While there have been many studies of code clone and their security issues on conventional software applications from various perspectives[9–17], there are few studies of code clones on VR software applications. We currently lack answers to many questions about code clone in VR software, such as whether VR software experiences code clone, how cloning levels differ among various VR software, which programming languages are more prone to VR code clone, and how code clone evolves throughout the lifecycle of VR software. These questions motivate us to conduct this empirical study on code clone in open-source VR software.

The contributions of this paper are as follows:

- We conduct an extensive quantitative empirical study of code clone in open-source VR software. To the best of our knowledge, no such studies exist in the literature that focus on code clone issues in VR software.
- We assemble a collection of 83 VR projects from GitHub, enabling a wide range of research into VR software.
- We propose to use four metrics that measure code clone from four angles (i.e., quantity, size, density, and diversity), allowing for a comprehensive evaluation when used together.
- We draw multiple insights from the experimental results, deepening our understanding of VR software.

2 Related Work

Researchers have dedicated efforts to investigating code clones. Al-Ekram et al. [7] analyzed source code clone across different software systems within the same domain to assess the extent of knowledge sharing. Juergens et al. [8] analyzed the impact of code clones on the correctness of software systems. Ekwa et al.[9] proposed Clonetracker, an Eclipse plug-in for tracking code clones in evolving software. Rajakumari et al. [10] introduced a novel conceptual framework designed to enhance software development and reuse by analyzing code clones. Mondal et al. [11] investigated the likelihood of errors in both method-level and fragment-level clones throughout the software's evolution processes, aiming to help manage code clones in software maintenance. Islam et al. [12] conducted an empirical comparative study on the differences and severity of security vulnerabilities between cloned and non-cloned code. Solanki et al. [13] found that code clones can increase system complexity, maintenance costs, and the potential for vulnerabilities. To detect code clones, Kamiya et al.[14] presented CCFinder, a token-based code clone detection tool for large scale source code. Martin et al.[15] introduced a deep learning based code clone detection tool. Cordy et al. [16] developed NiCad, a scalable and flexible clone detection tool. Feng et al. [17] proposed Nicad+, an improved version of NiCad, enhancing detection efficiency.

3 Dataset and Study Design

3.1 Dataset Construction

Despite the availability of multiple VR datasets from previous research [18–20], we create a dataset that is more precisely tailored to our research objectives, emphasizing popular projects that are more prone to cloning and compatible with the cutting-edge clone detection tool, NiCad. The generation of the dataset follows three phases, namely: project selection, analysis preparation, and manual inspection. Next, we describe each of these phases.

A . ***Phase 1: project selection.***
To build our dataset, we decide to collect a set of open-source VR projects from GitHub. We first search projects that match the keywords *"VR"* or *"Virtual Reality"*, getting about 134.2K repositories. To sieve out the noise (e.g., toy projects like homework assignments and poor-quality projects) that might distort the results, we adopt a stargazers-based classification following [21, 22], and only consider projects with GitHub stars greater or equal to two hundred (*stars* >= 200). This is reasonable because studies have shown that only a very small percentage of scored GitHub repositories contain engineered software projects [21]. Subsequently, we consider the status of the projects and further select projects that have updates after March 28th, 2016. This date has been chosen because it is the release day of the first commercial version of a computer-hooked HMD, marking the real arrival of VR technology. After the filtration (*stars* >= 200 & *pushed* >= 03/28/2016), we obtain 326 projects in total to be further handled.

B . ***Phase 2: Analysis preparation.***
Given that the open-source VR projects from GitHub are implemented on a range of platforms, we propose to identify the projects that can be analyzed with off-the-shelf clone detection tools. For this purpose, we use NiCad [16], one of the publicly available state-of-the-art code clone detection tools, to find clones in this study. Currently, the latest version is NiCad-6.2 [23], which handles a range of languages, including C, C#, Java, Python, PHP, Ruby, Swift, ATL, and WSDL. By applying this tool to the dataset obtained earlier, we extract 119 (out of 326) projects suitable for analysis.

C . ***Phase 3: Manual inspection.***
To ensure the precision of the dataset, we finally perform a manual check on the tags and "README.md" files for each project so as to retain only genuine VR applications. We eliminate the duplicate projects obtained from the query of *"VR"* and *"Virtual Reality"* respectively, and merge the two into a unified dataset. As a result of this phase, we finalize a sample of 83 VR applications.

Table 1 lists the results of each individual phase. Note that *Phase 2* and *Phase 3* are interchangeable. We choose this sequence to lessen the manual review effort.

Table 1: The amount of identified projects after each phase.

Keywords	Phase 1	Phase 2	Phase 3
VR	293	101	/
Virtual Reality	33	18	/
VR / Virtual Reality	326	119	83

3.2 Study Design

To conduct a quantitative analysis, we propose to use multiple metrics to evaluate the code clone level of open-source VR software. We first use two metrics to measure the absolute level of code clones in a VR project from two perspectives:

- NCF: This metric measures the total number of clone functions in a VR software. A function is deemed a clone function if there exists at least one other function in the software with a difference less than the dissimilarity threshold.
- NCC: This metric measures the number of clone classes in a VR software. A clone class is a group of two or more functions, where the difference between each two functions stays within the dissimilarity threshold.

We further use two metrics to measure the relative level of code clones in a VR project:

- rcf: This metric indicates the ratio between the number of clone functions (NCF) and the total number of functions (N) in a project, denoted by rcf = NCF/N.

Table 2: NiCad settings for code clone detection

Parameters	Target Clone Types					
	Type1	Type2	Type2c	Type3-1	Type3-2	Type3-2c
Dissimilarity Threshold	0.0	0.0	0.0	0.3	0.3	0.3
Identifier Renaming	none	blind	consistent	none	blind	consistent
Granularity	function	function	function	function	function	function

- rcc: This metric indicates the ratio between the number of clone classes (NCC) and the total number of functions (N) in a project, denoted by rcc = NCC/N.

4 Evaluation

4.1 Experiment Settings

In this paper, we use NiCad-6.2 for detecting code clones in the constructed dataset. NiCad can detect both exact and near-miss clones at the function or block level of granularity. The comparison uses a dissimilarity threshold to enable near-miss detection. For example, a threshold of 0.0 will find only exact clones, whereas a threshold of "0.1" allows for up to 10% difference. Besides, NiCad offers three different levels of detection for renamed clones (i.e., none, blind, consistent), handling identifiers in code fragments in three different ways. The parameter settings of NiCad are shown in Table 2, yielding a total of six types of code clones for detection. We consider code clones at the granularity of functions in order to obtain results with greater specificity. We set the dissimilarity threshold to "0" and "0.3" because "0" refers to the extreme scenario of exact sameness while "0.3" is the default value.

4.2 Experiment Results

In this section, we conduct empirical studies to explore five RQs, and analyze the experimental results to draw meaningful insights.

A . ***RQ1: Which projects are the most heavily cloned? And how do different detection settings influence the results?***

To address this RQ, we apply NiCad for code clone detection on the 83 VR software, considering six different types as outlined in Table 2. The top ten projects with the highest number of code clones are shown in Table 3, sorted by N (i.e., the total number of functions in a project) in descending order for better observation.

We first observe that projects with the largest number of functions are also those with the most code clones. The top three projects with the largest amount of code clones are "BlenderXR", "gpac", and "The-Seed-Link-Future". This suggests that VR project development, like conventional project development, experiences more code clone problems as the scale grows in most cases.

We further investigate the impact of using different detection types on the results. For each VR project, when comparing NCF across the 6 clone types, we find NCF(Type 3-2) > NCF(Type 3-2c) > NCF(Type 3-1), and NCF(Type 2) > NCF(Type 2c) > NCF(Type 1). This implies that the blind renaming approach always identifies the highest number of clone fragments, followed by the consistent renaming. These results are consistent with NiCad's theoretical expectations[16], where blind renaming alters all identifiers to "X", whereas consistent renaming assigns them as "X_n"(n is a sequence number). This makes sense because if a fragment is deemed similar when identifiers are renamed to "X_n", it will also be considered similar when all identifiers are renamed to "X". In addition, we notice that NCF(Type 3-x) > NCF(Type x) for x = 1, 2, 2c. This is straightforward because if a clone fragment satisfies a 0% dissimilarity threshold, it will naturally satisfy a 30% threshold. Consequently, it can be inferred that Type 3-2 is the most effective in detecting code clone functions, with its results covering those of the other types. On this basis, we decide to adopt Type 3-2 as the cloning type for the following studies.

Insight 1. *Most of the time, the larger the VR software size, the more code clones there are.*

Insight 2. *The impact of different cloning types on detection results is substantial. Both identifier renaming and dissimilarity threshold play crucial roles in determining the results.*

B . ***RQ2: How to choose the appropriate metrics for quantitative analysis of different purposes?***

To answer this RQ, we analyze the applicability of four metrics (i.e., NCF, NCC, rcf, rcc) as mentioned in Section 3.2.

The physical interpretation of NCF is straightforward; a larger value suggests a greater amount of cloning. However, the meaning of NCC (i.e., the number of clone classes) is less obvious. Take Type3-2 in Table 2 as an example. There is a large discrepancy between the NCF and NCC. The NCF ratio between Project 2 and Project 1 is about $2699/4448 \approx 60.7\%$, while the NCC ratio is much lower at $326/1226 \approx 26.6\%$. Conversely, between Project 4 and Project 3, the NCF ratio is $409/462 \approx 88.5\%$, with the NCC ratio even higher at $163/166 \approx 98.2\%$. To identify the cause, we study the correlation between the number of clone classes and their sizes. As shown in Table 4, we find that Project 2 has a noticeably higher number of big-size (>100) and mid-size ([11–100]) classes compared to Project 1, resulting in a significant drop in NCC. In contrast, Project 4 tends to have overall smaller class sizes than Project 3, causing the increase in NCC. Therefore, NCC can be seen as an indicator of code clone size; a lower value very likely implies the presence of large cloning classes.

To investigate the implications of rcf (i.e., the proportion of clone functions, measuring the density of function-level code clone) and rcc (i.e., the proportion of unique clone classes, measuring the diversity of code clone patterns), we plot how their values change across different projects, as shown in Fig. 1. We observe that rcf experiences substantial fluctuation, whereas rcc remains relatively stable. This suggests that the density of function-level code clone varies across different projects, whereas the diversity of code clone classes tends to stay stable. Additionally, we notice a lack of obvious correlation between rcf and rcc, indicating that density and diversity are not strongly linked.

Insight 3. *The four metrics we use (i.e.,* NCF, NCC, rcf, rcc*) evaluate code clone from four distinct perspectives (i.e., quantity, size, density, diversity), allowing for a comprehensive assessment when used together.*

C . ***RQ3:What programming languages are preferred by developers for VR open source software development? And which languages are more prone to generating code clones?***

Table 3: The quantity of detected code clones across six different cloning types.

PID	Project Name	Language	Type1		Type2		Type2c		Type3-1		Type3-2		Type3-2c		N
			NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	NCF	NCC	
1	BlenderXR	C;Python	358	175	1578	601	1486	567	1731	716	4448	1226	3112	889	27822
2	gpac	C	6	3	1421	259	1268	220	1195	312	2699	326	2089	278	10170
3	The-Seed-Link-Future	C#	44	19	231	84	223	85	201	86	462	166	295	112	8475
4	open-brush	C#	38	17	132	60	127	58	183	81	409	163	218	89	7548
5	UnityOculusAndroidVRBrowser	C#	36	15	156	53	150	55	107	47	255	86	177	65	3716
6	UnityPlugin	C#	12	6	53	21	52	21	60	24	113	43	85	33	2799
7	lovr	C	2	1	84	33	77	30	77	31	232	64	148	53	2173
8	UnityGameTemplate	C#	20	10	60	25	59	25	53	22	142	47	98	35	1981
9	ViveInputUtility-Unity	C#	20	10	62	27	62	27	69	32	125	49	84	36	1863
10	com.xrtk.core	C#	3	1	49	18	49	18	42	17	148	51	82	32	1607

Table 4: The distribution of the sizes of clone classes.

Interval / Project Name	2	[3,10]	[11, 100]	> 100
BlenderXR	898(73.2%)	304(24.8%)	23(1.9%)	1(0.1%)
gpac	207(63.5%)	100(30.7%)	13(4.0%)	6(1.8%)
The-Seed-Link-Future	122(73.5%)	39(23.5%)	5(3.0%)	0
open-brush	123(75.5%)	37(22.7%)	3(1.8%)	0

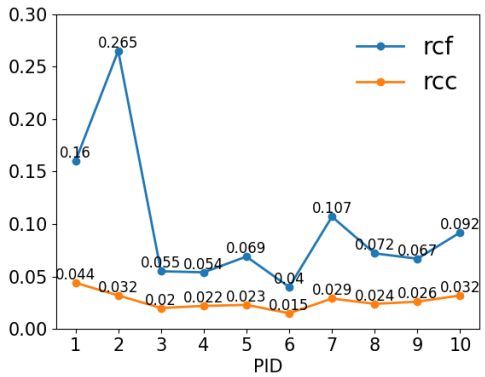


Figure 1: The changes of rcf and rcc across diverse projects.

To answer this RQ, we plot Fig. 2 to show the programming languages used by these software projects. In this context, the programming language mentioned for a project refers to the primary language used within the project.

We observe that C# is the most commonly used programming language in VR project, exceeding 50% of the total. This is most likely attributed to its close connection with the Unity game engine, which has emerged as one of the leading tools in VR development due to its ease of use and cross-platform features. Given that C# is the primary programming language for Unity, it's not surprising that it has become the most popular language in VR projects. Other languages used include Python, C, Java, and Swift, whereas PHP, Ruby, WSDL, and ATL are not employed. This is reasonable because web-oriented languages like PHP, Ruby and WSDL might not be ideal for graphics-intensive applications in terms of execution efficiency and performance, while ATL is mainly used for Windows programming. It should be noted that JavaScript is also a key language used in VR project development. It is not present because NiCad lacks support for detecting this language.

We further investigate which programming languages are more likely to result in code clones. To achieve this, we identify projects

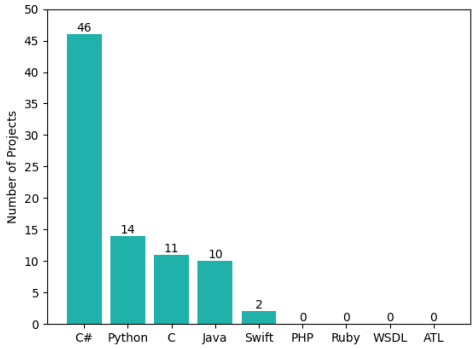


Figure 2: Programming languages used in VR projects.

with more than 10 clone classes (NCC >= 10) in terms of Clone Type 3-2, and analyze the programming languages employed by these projects. Table 5 shows the number of projects found at different cloning levels using various languages. We observe that C# leads to the highest number (18) of projects with more than 10 clone classes, followed by C (5). This makes sense since C# is the most widely used language in VR software development. However, with regard to the cloning ratio, C(5/11 ≈ 45.5%) is higher than C# (18/46 ≈ 39.1%), meaning that C language is more likely to trigger cloning issues. This can be justified as C language underpins many low-level systems and graphics libraries(e.g., OpenGL), with abundant resources facilitating code reuse.

Insight 4. *C# is the most frequently used programming language in open-source VR projects, whereas C is more likely to cause cloning issues.*

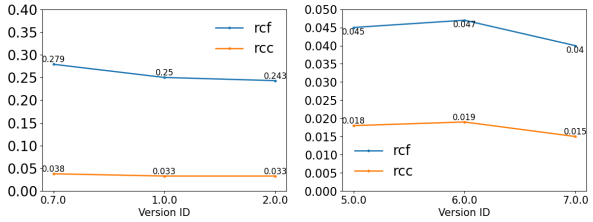
Table 5: The distribution of projects across different cloning intervals with regard to NCC using various languages.

NCC / Language	[10, 30]	[31, 50]	[51, 100]	> 100	Sum
Java	1	0	0	0	1 (out of 10)
Python	3	0	0	1	4 (out of 14)
C	1	1	1	2	5 (out of 11)
C#	7	7	2	2	18 (out of 46)

D. RQ4: How does intra-version code clone evolve across different versions of a VR project?

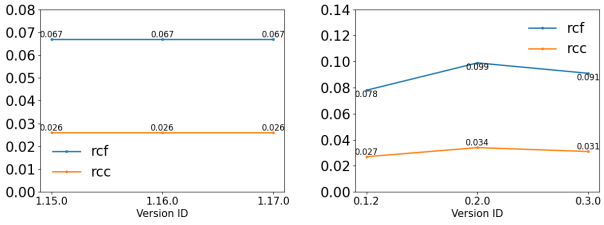
To address this RQ, we break it down into three sub-RQs: how intra-version code clones evolve when major version number changes

(e.g., 0.x.x, 1.x.x, 2.x.x), secondary version number changes (e.g., 0.1.x, 0.2.x, 0.3.x), and ending version number changes (e.g., 0.1.1, 0.1.2, 0.1.3). For analyzing each sub-RQ, we choose two distinct projects, focusing on their latest three eligible versions. It is worth noting that we select these two projects because their frequent releases facilitate the analysis.



(a) gpac (b) UnityPlugin

Figure 3: Detection results of intra-version clones across multiple versions when major version number changes.



(a) ViveInputUtility-Unity (b) com.xrtk.core

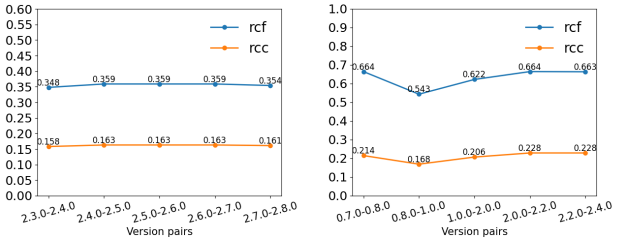
Figure 4: Detection results of intra-version clones across multiple versions when secondary version number changes.

Figure 3(a) and Figure 3(b) plot rcf, rcc as the major version number of the VR project "gpac" and "UnityPlugin" increase, respectively. We notice that in the project "gpac", rcf varies by up to 0.036 and rcc by up to 0.005 across three main versions. In project "UnityPlugin," rcf differs by up to 0.007, and rcc by up to 0.004. This suggests that the number of code clones, measured by rcf and rcc, stays relatively stable when the major version changes for VR projects.

Figure 4(a) and Figure 4(b) display rcf, rcc as the secondary version number of the VR project "ViveInputUtility-Unity" and "com.xrtk.core" increase, respectively. We notice that in the project "ViveInputUtility-Unity", rcf and rcc remain unchanged across three main versions. In project "UnityPlugin," rcf differs by up to 0.021, and rcc by up to 0.007. This implies that the number of code clones, measured by rcf and rcc, also stays relatively stable when the secondary version changes for VR projects. Sometimes, there isn't any change at all.

Note that we do not include the figures when the ending version number changes because the variation among different versions is negligible.

Insight 5. *Though the degree of code clone may vary from project to project, within the same project, the level of intra-version code clone remains relatively stable.*



(a) open-brush (b) gpac

Figure 5: Detection results of inter-version code clones on five adjacent version pairs of two projects.

E. RQ5: How does inter-version code clone evolve across different versions of a VR project?

To explore whether different versions of the same project clone from each other, we execute code clone detection on six sequential versions of each project, with the detection carried out between each pair of adjacent versions, yielding five data sets per project. We plot the experimental results of two projects in Figure 5.

We notice that in the "open-brush" project, the values of rcc and rcf are within the ranges of [0.158,0.163] and [0.348,0.359], while in the "gpac" project, rcc and rcf values are within the ranges of [0.168,0.228] and [0.543,0.664]. This implies that although the degree of inter-version code clone varies across different projects, it remains relatively high overall, emphasizing the prevalence of inter-version code clone in VR software development. While code reuse can enhance development efficiency, it carries the risk of propagating hidden vulnerabilities to subsequent versions.

Insight 6. *Inter-version code clone is prevalent in VR software development, posing a high security risk as a vulnerability in cloned code could affect many versions.*

5 Conclusions and Future Work

In this work, we present a quantitative empirical study of code clone on open source VR projects. We first compile a dataset of 83 VR projects from GitHub for study, adhering to rigorous criteria. Then, we propose to use a suite of metrics for measuring code clone from diverse perspectives. Finally, we carry out empirical experiments on five carefully designed research questions to capture the landscape of VR software code clone. One important finding is that code clone is widespread in VR software, both within and between versions. This poses a significant potential risk to VR software security, warranting further attention and research.

Future work on this topic can proceed in several directions. One clear advancement is comparing cloning behavior in VR and non-VR software to determine if the findings are unique to VR. If so, a detailed study of the distinctions between VR and traditional code clone is necessary, as well as an exploration of particular types of VR clones that traditional detection tools cannot detect, to support the development of clone detection tools tailored for VR. Additionally, studying VR-specific clone characteristics, such as 3D model clones, model-related code clones, and clones in meta files, is important.

References

[1] Christoph Anthes, Rubén Jesús García-Hernández, Markus Wiedemann, and Dieter Kranzlmüller. State of the art of virtual reality technology. In *2016 IEEE aerospace conference*, pages 1–19. IEEE, 2016.

[2] Leif P Berg and Judy M Vance. Industry use of virtual reality in product design and manufacturing: a survey. *Virtual reality*, 21:1–17, 2017.

[3] Mordor intelligence report on virtual reality market. https://www.mordorintelligence.com/industry-reports/virtual-reality-market. (Accessed on 08/09/2024).

[4] Virtual reality statistics: The ultimate list in 2024. https://academyofanimatedart.com/virtual-reality-statistics/. (Accessed on 08/10/2024).

[5] Hanyang Guo, Hong-Ning Dai, Xiapu Luo, Zibin Zheng, Gengyang Xu, and Fengliang He. An empirical study on oculus virtual reality applications: Security and privacy perspectives. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[6] Alberto Giaretta. Security and privacy in virtual reality–a literature survey. *arXiv preprint arXiv:2205.00208*, 2022.

[7] Raihan Al-Ekram, Cory Kapser, Richard Holt, and Michael Godfrey. Cloning by accident: an empirical study of source code cloning across software systems. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 10–pp. IEEE, 2005.

[8] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *2009 IEEE 31st International Conference on Software Engineering*, pages 485–495. IEEE, 2009.

[9] Ekwa Duala-Ekoko and Martin P Robillard. Clonetracker: tool support for code clone management. In *Proceedings of the 30th international conference on Software engineering*, pages 843–846, 2008.

[10] Kavitha Esther Rajakumari. Towards a novel conceptual framework for analyzing code clones to assist in software development and software reuse. In *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 105–111. IEEE, 2020.

[11] Smritikana Mondal, Manishankar Mondal, and Rameswar Debnath. Granularity-based comparison of the bug-proneness of code clones. In *2023 IEEE 17th International Workshop on Software Clones (IWSC)*, pages 8–14. IEEE, 2023.

[12] Md Rakibul Islam, Minhaz F Zibran, and Aayush Nagpal. Security vulnerabilities in categories of clones and non-cloned code: An empirical study. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 20–29. IEEE, 2017.

[13] Kamna Solanki and Sunayna Kumari. Comparative study of software clone detection techniques. In *2016 Management and Innovation Technology International Conference (MITicon)*, pages MIT–152. IEEE, 2016.

[14] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE transactions on software engineering*, 28(7):654–670, 2002.

[15] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, pages 87–98, 2016.

[16] James R Cordy and Chanchal K Roy. The nicad clone detector. In *2011 IEEE 19th international conference on program comprehension*, pages 219–220. IEEE, 2011.

[17] Chenhui Feng, Tao Wang, Jinze Liu, Yang Zhang, Kele Xu, and Yijie Wang. Nicad+: Speeding the detecting process of nicad. In *2020 IEEE international conference on service oriented systems engineering (SOSE)*, pages 103–110. IEEE, 2020.

[18] Dhia Elhaq Rzig, Nafees Iqbal, Isabella Attisano, Xue Qin, and Foyzul Hassan. Virtual reality (vr) automated testing in the wild: A case study on unity-based vr applications. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1269–1281, 2023.

[19] Fariha Nusrat, Foyzul Hassan, Hao Zhong, and Xiaoyin Wang. How developers optimize virtual reality applications: A study of optimization commits in open source unity projects. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 473–485. IEEE, 2021.

[20] Irving Rodriguez and Xiaoyin Wang. An empirical study of open source virtual reality software projects. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 474–475. IEEE, 2017.

[21] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. Curating github for engineered software projects. *Empirical Software Engineering*, 22:3219–3253, 2017.

[22] Naoures Ghrairi, Sègla Kpodjedo, Amine Barrak, Fabio Petrillo, and Foutse Khomh. The state of practice on virtual reality (vr) applications: An exploratory study on github and stack overflow. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 356–366. IEEE, 2018.

[23] Nicad clone detector. http://www.txl.ca/txl-nicaddownload.html. (Accessed on 08/10/2024).