An Exploratory Study of Bugs in Extended Reality Applications on the Web

Shuqing Li*, Yechang Wu*, Yi Liu*, Dinghua Wang*†, Ming Wen‡, Yida Tao§, Yulei Sui†, and Yepang Liu*
*Dept. of Comp. Sci. and Engr., Southern University of Science and Technology, Shenzhen, China.
{lisq2017,11711918,11610522,11960002}@mail.sustech.edu.cn, liuyp1@sustech.edu.cn
†Faculty of Engr. and IT, University of Technology Sydney, Sydney, Australia.
13284361@student.uts.edu.au, yulei.sui@uts.edu.au
‡School of Cyber Sci. and Engr., Huazhong University of Science and Technology, Wuhan, China. mwenaa@hust.edu.cn
§College of Comp. Sci. and Soft. Engr., Shenzhen University, Shenzhen, China. yidatao@szu.edu.cn

***Abstract*—Extended Reality (XR) technologies are becoming increasingly popular in recent years. To help developers deploy XR applications on the Web, W3C released the WebXR Device API in 2019, which enable users to interact with browsers using XR devices. Given the convenience brought by WebXR, a growing number of WebXR projects have been deployed in practice. However, many WebXR applications are insufficiently tested before being released. They suffer from various bugs that can degrade user experience or cause undesirable consequences. Yet, the community has limited understanding towards the bugs in the WebXR ecosystem, which impedes the advance of techniques for assuring the reliability of WebXR applications. To bridge this gap, we conducted the first empirical study of WebXR bugs. We collected 368 real bugs from 33 WebXR projects hosted on GitHub. Via a seven-round manual analysis of these bugs, we built a taxonomy of WebXR bugs according to their symptoms and root causes. Furthermore, to understand the uniqueness of WebXR bugs, we compared them with bugs in conventional JavaScript programs and web applications. We believe that our findings can inspire future researches on relevant topics and we released our bug dataset to facilitate follow-up studies.**

***Index Terms*—WebXR, empirical study, bug taxonomy**

I. Introduction

How to design better mechanisms for humans, computers, and the environment to perceive and interact with each other has long been an important research topic, especially in the field of *human-computer interaction* (HCI). In recent years, several emerging technologies have become increasingly popular as they provide new mechanisms for users to embrace immersive experiences while interacting with computers and the environment [1], [2]. These technologies include: (1) setting up an artificial world for simulated user experiences (Virtual Reality, VR), (2) adding digital information or graphics on top of real-world scenes (Augmented Reality, AR), and (3) creating virtual objects in the real world and allowing them to interact with physical objects (Mixed Reality, MR) [3]. They are collectively known as the Extended Reality (XR) technologies. Users can interact with applications powered by XR technologies (XR applications) via special devices such as head-mounted displays, hand-held controllers, and locators.

*The first three authors contributed equally to this work. Yepang Liu is the corresponding author.

XR applications have been widely deployed for various purposes, such as entertainment [4], health care [5], skill training [6], and vehicle design [7]. Many of the scenarios require XR applications to be highly reliable. For instance, doctors may conduct physical treatment [8] with the help of XR technologies. XR applications to be used in such a scenario should be thoroughly tested before being deployed.

Conventional XR applications require special devices and platforms. For example, some XR applications can only be downloaded from a certain app store and used on the devices of a particular brand. This increases the barrier for users and developers to access XR applications. To address this problem, the World Wide Web Consortium (W3C) has released a group of standards, collectively known as the *WebXR Device API* (WebXR for short), to enable XR applications to be deployed on the Web [9], [10]. WebXR is built on top of *WebGL* [11], a JavaScript API to display 2D or 3D graphics. It provides developers with the APIs to: (1) recognize and connect to XR devices, (2) obtain inputs from XR devices, (3) render scenes to the displaying equipment with a certain frequency (i.e., *frame rate*) or send output to special devices, and so on [12]. This greatly eases the development of XR applications. Besides facilitating application development, WebXR also brings convenience to users. With WebXR, users can access XR applications in any mainstream browsers (e.g., Chrome and Firefox) as well as on any devices that are equipped with basic web browsing capabilities. Given such benefits for both developers and users, WebXR has become increasingly popular recently [13], [14]. For example, *A-Frame* and *Babylon.js*, two open-source frameworks for building WebXR applications, have received thousands of downloads monthly on *npmjs.com* [15] and over 10k stars on GitHub [16].

Despite the growing popularity and impact of WebXR, many WebXR applications may be insufficiently tested and shipped with unexpected bugs. For example, we observed hundreds of open issues waiting to be resolved in the GitHub repository of A-Frame [17]. These bugs can significantly degrade user experience or cause undesirable consequences (see Section IV). Unfortunately, since WebXR is still in its infancy, there is limited understanding of the bugs in the WebXR ecosystem (WebXR bugs for short), including WebXR frameworks, libraries, and applications. In particular, there are no effective techniques for developers to detect, diagnose, and fix WebXR bugs.

In this work, we conducted the first empirical study of WebXR bugs to understand their characteristics. For the study, we collected 368 real bugs by analyzing the issue reports and release notes of 33 WebXR projects hosted on GitHub. Via a seven-round manual analysis of these bugs, we learned the common symptoms and causes of WebXR bugs and built a bug taxonomy accordingly (see Sections IV and V). This analysis also led to several interesting findings. For example, we observed that WebXR applications, built on top of various frameworks or libraries, can run on diversified platforms with different browsers, XR devices, operating systems, which may not be fully compatible and interoperable with each other. Due to such fragmentation in the ecosystem, compatibility issues are widely spreading among the WebXR ecosystem. In addition, due to the complicated human-computer interaction mechanisms, the input space of WebXR applications can be much larger than that of traditional software. As many WebXR bugs require special user interactions to manifest, the large input space brings challenges to automated bug detection.

To further understand the uniqueness of WebXR bugs, we compared them with bugs in conventional JavaScript programs and web applications. We did not compare with bugs in traditional XR applications (e.g., those deployed on specific devices/platforms) because we did not find any existing studies on bugs of such applications. Via the comparison, we identified several types of bugs that are specific to WebXR applications, which are worth further studies (see Section VI). To summarize, our work makes the following contributions:

- To the best of our knowledge, we conducted the first empirical study of bugs in real-world WebXR projects and built the bug taxonomy. Our findings can help better understand WebXR bugs and shed light on future research.
- We compared WebXR bugs and bugs that often occur in conventional JavaScript programs and web applications. This comparison reveals the uniqueness of WebXR bugs.
- To facilitate follow-up studies, we have released our bug dataset and materials of our manual analysis process at the site https://sites.google.com/view/webxr-bug-study/ and Zenodo (https://doi.org/10.5281/zenodo.3992105).

II. Background

A. WebXR Device API

How to deploy XR applications on the Web has been discussed for years. In the beginning, different organizations and individual developers have their own implementations to support AR/VR on the Web. This induced huge challenges to guarantee the compatibility of different applications. To address the problem, Mozilla's VR team and Google's Chrome team released a specification known as *WebVR* in March 2016 to help developers build VR applications on the Web [18], [19]. Since then, the mainstream browsers began to support WebVR one after another. Multiple WebVR frameworks such as *A-Frame* and *ReactVR* also emerged and quickly received plenty of attention [20]. However, WebVR could not help developers build and deploy AR or MR applications on the Web. To bridge this gap, several Mozilla developers formed a team, launched the Mixed Reality program, and started a draft proposal for new standards to support all XR content on the Web. This team, which was later known as the *Immersive Web Community Group* of W3C, released the first public working draft of WebXR Device API in February 2019 [21]. Since then, WebVR has been deprecated [22]. Most of the once-popular WebVR projects have migrated to WebXR (e.g., *A-Frame*) but might still support WebVR for backward compatibility.

WebXR combines all XR technologies, standardizes and integrates them as a whole ecosystem. The standards are under active development and revisions. So far, W3C has released four public working drafts of WebXR [21], [23]–[26], and the API has been widely used in various real-world projects.

B. Interacting with WebXR Applications

For conventional web applications, users usually interact with them via devices like mice, keyboards and cameras (usually for mobile applications). Besides these traditional devices, WebXR supports more devices to provide immersive user experiences. For ease of presentation, we refer to such special devices as ***XR devices***, which include but are not limited to the following ones [12], [24]:

- ***Head-mounted displays*** (HMD, also known as *headsets*), which play the computer-generated imagery and sound.
- ***Handheld controllers*** (also known as *gamepads*), which are used to control WebXR applications. Typically, there is a pair of controllers, one for each hand. Users can interact with WebXR applications by pressing buttons, altering the directions they are facing, and so on.
- ***Position tracking equipment*** (also known as *sensors* or *locator devices*), which can locate the relative position of the user. Note that not every XR device has position tracking equipment. Devices with three degrees of freedom (DoF) can only track users' head rotations around three different axes but do not support position tracking. Devices with six DoFs support position tracking and are able to recognize the up-down, left-right, forward-backward movement of users.

Generally, WebXR applications can be accessed in three different modes, namely, **non-XR (desktop) mode**, **full-screen mode**, and **XR mode**. Non-XR mode refers to the situation, where the WebXR applications work as embedded components of web pages. The users can make attempts to activate the XR mode from the non-XR mode. If the browser fails to detect or bind to XR devices, the applications will fallback to the full-screen mode, where the applications' UI is displayed using the whole screen.[1] Otherwise, the applications will switch to XR mode successfully, where users can interact with the WebXR applications using their body languages (e.g., walking in some

[1]This is the usual case for WebVR applications. Some of WebXR applications don't use exactly the whole screen.

directions, pressing buttons on the controllers, and turning their heads around). The XR devices will trace users' motions, generate corresponding input sequences, and send them to the browser. With WebXR, the browsers can recognize hardware inputs, create and dispatch structured inputs to the relevant application. Then the application can handle the user inputs and control the XR devices to display new scenes or play sounds.

III. Methodology

A. Research Questions

Our empirical study explores three research questions:

- **RQ1 (Bug types and symptoms):** What are the common types of WebXR bugs in real-world projects? What are the symptoms of these bugs?
- **RQ2 (Root causes):** What are the common root causes of WebXR bugs?
- **RQ3 (Uniqueness):** How do WebXR bugs differ from bugs in conventional JavaScript programs and web applications?

RQ1 and RQ2 focus on studying bug patterns. The symptoms and root causes can help us understand the flaws in WebXR projects. The causal relationships between symptoms and root causes are useful for guiding the design of testing methods in future research. To answer these two research questions, we first collected a large dataset of real bugs and their relevant information from open-source WebXR projects, and then manually analyzed each bug with the aim to construct a bug taxonomy and derive common and meaningful characteristics of WebXR bugs.

RQ3 concentrates on the differences between WebXR bugs and traditional software bugs to highlight the originality and potential significance of our empirical analysis. To answer this research question, we selected existing studies [38]–[40] on bugs in JavaScript programs and web applications for comparison. JavaScript programs are compared since WebXR applications are mostly written in JavaScript and web applications are compared since WebXR applications are deployed on the web and can be considered as a special type of web applications. We did not compare with traditional XR applications because we did not find empirical studies on bugs in such applications.

B. Data Collection

Our data collection process includes three steps:

1) ***Project Sampling****:* We collected WebXR projects from GitHub, which is the most popular open-source project hosting site. Specifically, we used the GitHub search engine to search for projects with the keywords of *WebXR*, *WebVR*, or *WebAR*. Note that using *WebVR* and *WebAR* as keywords might retrieve projects with old or non-standard specifications. Yet, as explained in Section II-A, since WebXR superseded WebVR, more and more old WebVR and WebAR projects have migrated to the new standards but with their descriptions unchanged. Hence, we chose these three keywords and used them collectively to obtain more complete results.

We selected two sets of projects from the search results: (1) the top 50 projects with the highest number of stars and (2) the top 50 projects according to the last modified time. We adopted these criteria because the subjects selected in this way enable us to find the latest and relevant bugs in popular projects. To reduce the risk of including old projects that follow the outdated WebVR and WebAR standards, we manually inspected the selected projects to check whether they are WebXR projects. Specifically, for each project's GitHub repository, we first checked: (1) if the README and documentation contain descriptions about which standard the project used and (2) if the repository depends on popular WebXR frameworks. We then used a browser extension called *WebXR API Emulator* [41] to test each project without using actual XR devices. Finally, we confirmed that 33 of the above selected 78[2] projects are indeed WebXR projects and we used them as subjects in subsequent analyses.

Note that our work focuses on investigating bugs in the entire ecosystem of WebXR and we aim to understand as many types of bugs as possible in this new domain from a general aspect. Due to this reason, we did not segregate the WebXR frameworks (or libraries) from WebXR applications when selecting projects.

2) ***Collecting Bugs****:* We collected bugs from the issue tracking systems and release notes of the 33 WebXR projects. We explain the process below.

First, from the **issue tracking systems** of the 33 projects, we collected the issues using the following three criteria:

- *The issue is closed*: As issues are often closed after they have been successfully resolved, we are more likely to identify the precise root causes from closed issues.
- *The issue is linked to patches*: This means that the issue has been fixed and the patches have been merged into the code repository. These patches provide rich information to help understand the bugs.
- *The fixing patch(es) should contain source code changes*: This helps filter out patches that contain documentation-only changes (e.g., updating README). The purpose of these changes is not to fix bugs and thus the issue is not relevant.

It is worth mentioning that issues of the selected projects rarely have explicit "bug fixing" or related labels. Consequently, the issues extracted using the above criteria might also be application-specific enhancements or feature requests that are not relevant to our research questions. Hence, we manually inspected all of the extracted issues to ensure that they indeed discuss WebXR bugs. Finally, we identified 171 WebXR bugs from the issue tracking systems.

Second, **release notes** summarize the most crucial changes between releases and typically provide three types of information: *changes*, *fixes*, and *enhancements*. Specifically, in the *fixes* part, developers would briefly describe the bugs and link them to the pull requests (or commits) that contain the fixes. We leveraged this information and collected WebXR bugs by

[2]The number is not equal to 100 because the search results have overlaps.

TABLE I: WebXR projects used in our empirical study

Project Name	Stars	Releases	Commits	Files	Lines of Code	Closed Issues	WebXR Bugs
jeromeetienne/AR.js[1][27]	13,260	37	1,156	340	487,356	168	2
aframevr/aframe [17]	10,096	25	5,453	450	144,570	398	227
BabylonJS/Babylon.js [28]	9,496	173	19,495	2,000	2,213,376	734	18
playcanvas/engine [29]	4,563	671	8,117	587	78,449	95	29
GoogleWebComponents/model-viewer[2][30]	694	20	447	190	27,882	200	38
donmccurdy/aframe-extras [31]	534	122	605	96	35,507	62	4
mozilla/hubs [32]	262	0	6,389	435	65,673	361	37
donmccurdy/aframe-physics-system [33]	236	27	140	51	24,080	28	1
mozilla-mobile/webxr-ios [34]	120	0	591	170	44,501	68	3
immersive-web/webxr-polyfill [35]	109	13	55	70	18,607	18	3
supermedium/moonrider [36]	58	0	324	166	57,916	40	5
engaging-computing/MYR [37]	7	21	1,117	71	8,511	76	1

[1] The newly maintained versions have been moved to https://github.com/AR-js-org/AR.js now.
[2] This repository has been entirely moved to https://github.com/google/model-viewer.

processing the *fixes* information of every release note of the 33 projects. Similarly, we filtered out bugs that are not linked with patches. After removing duplicates with bugs collected from issue tracking systems, we identified 197 WebXR bugs from the release notes.

In total, we collected 368 bugs from 12 of the 33 WebXR projects. There were no WebXR bugs found in the other 21 projects, following our criteria. Table I gives the statistics of the bugs and projects.

3) ***Collecting Bug Details****:* To facilitate our manual analysis of bugs, we collected useful information for each of the 368 WebXR bugs, including the URLs of the issue and patch(es), the creation time of the issue, the merge time of the patch(es), and the release version (only for bugs collected through release notes). Such information allows us to access the patches and understand the timeline of the bug-fixing activities. We also collected all the textual contents from the issue threads and pull requests of these bugs. The textual contents include descriptions of the bugs or pull requests, as well as the follow-up comments, some of which also provide the setup information (e.g., OS, device, browser, and framework version), steps to reproduce, and proof-of-concept code snippets. These types of information are essential for us to reliably and effectively study the symptoms and root causes of WebXR bugs.

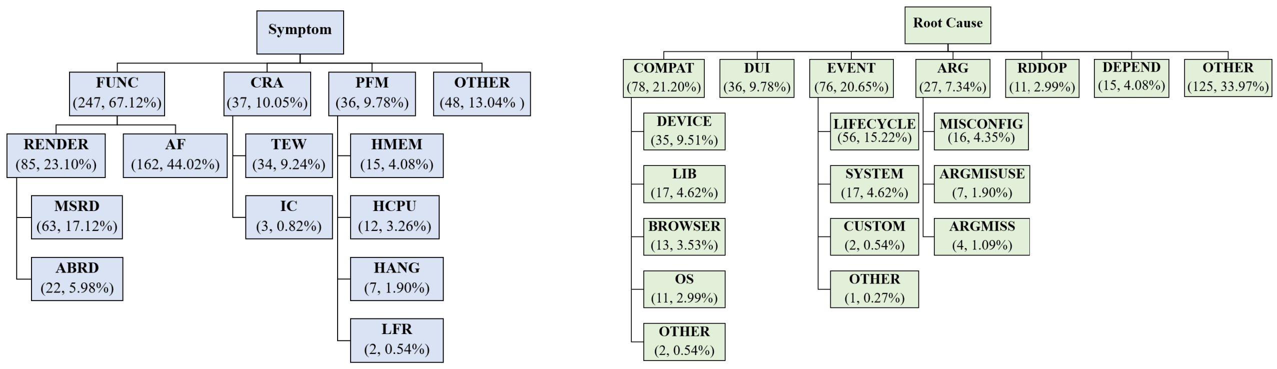
C. Data Analysis

To characterize the symptoms and root causes of the 368 WebXR bugs, we analyzed the data following an open coding procedure [42], which is widely used for qualitative data analysis. Specifically, we conducted an iterative manual labeling process involving three evaluators (i.e., co-authors of the paper), who all have several years of web development experience. In the first iteration, we determined an initial draft of bug taxonomy based on the intuitive knowledge acquired from the data collection process. We also discussed an initial labeling strategy with respect to the symptoms and root causes. In the second iteration, every evaluator independently labeled all of the collected WebXR bugs based on the taxonomy and labeling strategy discussed before, by carefully inspecting the issue threads, pull requests, and the committed patches of every bug. The three evaluators then gathered together to compare and discuss the results, with the purpose of clarifying the descriptions and boundaries of different categories, adding and removing categories, adjusting the hierarchical structures of categories, and adopting a much more clear-cut labeling strategy. Then, in the subsequent iterations, the three evaluators conducted the labeling process again using the adjusted taxonomy and strategy from the prior iteration. The evaluators reached a consensus on the taxonomy of both the bug symptoms and root causes after the seventh iteration.

The resulting taxonomy is shown in Figure 1 and explained in detail in Sections IV and V. Basically, each bug is labeled as exactly one leaf category of the symptom taxonomy and one leaf category of the root cause taxonomy. For bugs with multiple symptoms or root causes, we classified them into one category using the following strategy. First, for bugs with multiple symptoms, we chose the most obvious symptom from the end-users' point of view. To this end, we considered ourselves as end-users that prefer to focus on the graphical user interface rather than developer tools such as the browser consoles. If there were still multiple symptoms, we selected the symptom that appeared the first. Second, for bugs with multiple root causes, we chose the one from which the bug initially originates. For example, if a bug occurred due to argument misuse, and the argument misuse is actually resulted from an API change of a third-party library, then we consider the root cause to be *incompatible runtime environment* rather than *argument misuse*. Finally, bugs with too complicated patches or too limited diagnostic information were labeled as *others*, as we were unable to assign them to a matching category with enough confidence.

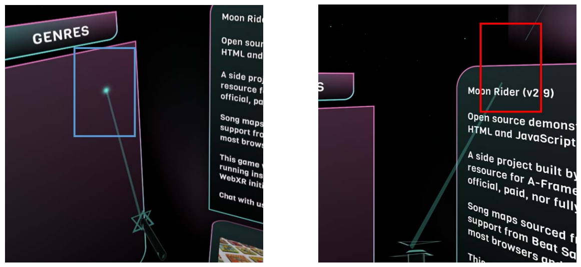
IV. RQ1: Bug Types and Symptoms

We studied each of the 368 WebXR bugs and classified the bugs according to their symptoms. We found that 320 of our studied bugs can be classified into three types: (1) functional issues, (2) crashing issues, and (3) performance issues. The remaining 48 bugs are either rarely observed or difficult to understand. In this section, we present the three major types of WebXR bugs in detail.



(a) Symptom (b) Root Cause

Fig. 1: The taxonomy of WebXR bugs with respect to symptoms (a) and root causes (b).



(a) Normal (b) Faulty

Fig. 2: An example of application-specific functional issues (the ray penetrates the right menu but not other menus).

A. *Functional Issues (FUNC)*

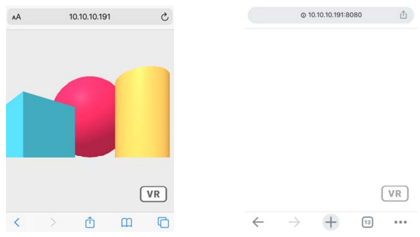
As shown in Figure 1(a), functional issues are the most common type of WebXR bugs (67.12%). They can be further classified into the following two subtypes:

Application-Specific Functional Issues (AF): 162 of the 247 functional issues have various application-specific symptoms. For example, *moonrider* is a music playing game. As shown in Figure 2, at the starting scene, the ray collides with the center and left menus but penetrates the right menu (issue #14 [43]).[3] This could mislead users to interact with the right menu in a different way. As another example, when users exit the VR mode, the application *Mozilla Hubs* stops playing sound (issue #1099 [44]). Such unexpected behaviors are often caused by improper lifecycle event handling and erroneous design of interactive logic, which we will discuss later.

Finding 1: *The majority (247 of 368) of our studied WebXR bugs are functional issues, around 66% of which have application-specific symptoms.*

Rendering Issues (RENDER): 85 of the 247 functional issues are related to scene rendering. These rendering issues can be further classified into: (1) *misrendering of objects* (MSRD, 63 issues) and (2) *absence of objects* (ABRD, 22

[3]To avoid copyright issues, we reproduced the WebXR bugs and took screenshots to illustrate the bug examples in Sections IV and V. The original figures posted by issue reporters can be accessed via the issue links.



(a) On Safari (b) On Chrome

Fig. 3: An example of absence of objects (scenes can be rendered on Safari but not on Chrome for iOS devices).

issues). WebXR applications render many objects and utilize various animations to interact with users. The absence or misrendering of objects will seriously affect the functionality of the applications and degrade user experiences. For example, as shown in Figure 3, applications using *A-Frame* can render scenes correctly on Safari but not on Chrome for iOS devices (issue #3846 [45]). Figure 4 gives an example of misrendered objects. By using *A-Frame*, the text of an application becomes blurred when the opacity is set to a value that is smaller than a specific threshold (issue #3557 [45]).

Finding 2: *Misrendering and absence of objects are common functional issues in WebXR projects.*

B. *Crashing Issues (CRA)*

Crashing issues are also common. There are 37 such issues in our dataset, three of which caused *immediate application crashes* (IC). For example, Google's *model-viewer* crashed when users imported external models with third-party libraries enabled (issue #742 [46]). The remaining 34 issues caused the applications to *throw exceptions in the console with Error or Warning* (TEW). For instance, the third-party rendering library *Three.js* [47] will mock WebXR APIs if the browsers do not support WebXR. However, due to a bug (issue #3672 [45]), applications using *Three.js* will throw exceptions when running in browsers with native WebXR support.

Finding 3: *Around 10% of our studied WebXR bugs caused runtime exceptions or immediate application crashes.*

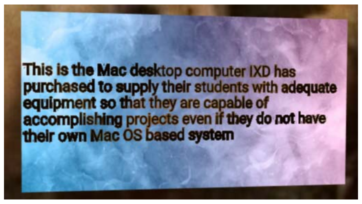
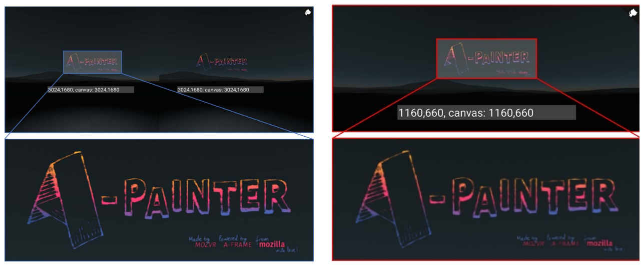

This is the Mac desktop computer IXD has purchased to supply their students with adequate equipment so that they are capable of accomplishing projects even if they do not have their own Mac OS based system

Fig. 4: An example of misrendering of objects (blurred text).


3024,1680, canvas: 3024,1680
3024,1680, canvas: 3024,1680
A-PAINTER
1160,660, canvas: 1160,660
A-PAINTER

(a) In VR mode (b) After exiting VR mode

Fig. 5: An example of resolution issues (the application's resolution drops after exiting VR mode).

C. *Performance Issues (PFM)*

Besides functional and crashing issues, performance issues are also common in WebXR projects. In total, we observed 36 performance issues in our dataset. The issues can be further classified into the following four subtypes according to their different symptoms:

High Memory Consumption (HMEM): 15 performance issues caused unexpectedly high memory consumption. For example, due to an issue in *playcanvas engine*, the memory usage keeps growing when games built on the engine recreate specific entities with customized assets (issue #1299 [48]).

High CPU Utilization (HCPU): 12 performance issues caused unexpectedly high CPU utilization and slowed down the applications. For example, an issue in *playcanvas engine* caused the floating point rendering to be slow on Windows 10 devices and smartphones (issue #1260 [48]).

Abnormal Hanging of Applications (HANG): Severe performance issues may cause an application to stop responding to user actions (i.e., the application hangs). We observed seven such issues in our dataset. For instance, due to an issue in *A-Frame* (issue #3617 [45]), the scenes of an application froze and the gyro stopped functioning in Magic Window Mode. Such issues are annoying since users will be forced to kill and restart the application to continue using it.

Low Frame Rate or Resolution (LFR): Besides excessive resource consumption, performance issues can also reduce frame rate or resolution, thus compromising user experience. Figure 5 gives an example. Due to an issue in *A-Frame* (issue #3592 [45]), the application's resolution drops from 1680p to 660p after users exit the VR mode on mobile Chrome.

Finding 4: *Performance issues in WebXR projects have various symptoms including high memory consumption, high CPU utilization, abnormal hanging of applications, and low frame rate or resolution.*

V. RQ2: Root Causes

Via analyzing our collected data, we observed six major root causes of WebXR bugs, including: (1) incompatible runtime environment, (2) event handling mistakes, (3) improper handling of diversified user interaction mechanisms, (4) wrong arguments, (5) buggy dependencies, and (6) redundant operations. In this section, we present each of them in detail.

A. *Incompatible Runtime Environment (COMPAT)*

As shown in Figure 1(b), 78 (21.20%) WebXR bugs are caused by *incompatible runtime environment*, including the incompatibility of different device models, third-party libraries and their APIs, browsers, and operating systems. Below we present examples of each type.

Incompatible Device Models (DEVICE): Oculus Go and Gear VR are two brands of XR devices. Oculus Go controllers identified themselves to browsers as *Gear VR* in previous time but changed to *Oculus Go* recently, which makes the controllers stop working on *A-Frame* based applications (issue #3390 [45]). We observed 35 such issues in our dataset.

Incompatible Third-party Libraries and APIs (LIB): *A-Frame* provides a panel to show the status of running applications. However, since a field name of *Three.js* (one of *A-Frame*'s dependencies) changed from *faces* to *triangles* in the newer version, the *faces* information on the status panel is shown as NaN (issue #3573 [45]). This issue was fixed by simply updating that field name as shown in Listing 1. There are 17 such issues in our dataset.

Listing 1: The fix of a bug caused by 3rd-party library evolution

```
- 'renderer.info.render.faces': {
-    caption: 'Faces',
+ 'renderer.info.render.triangles': {
+    caption: 'Triangles',
     over: 1000
  },
```

Incompatible Browsers (BROWSER): Users reported that the "View in AR" button of Google's *model-viewer* does not work as expected on certain versions of the Chrome browser (issue #693 [46]). For some models, clicking the button makes certain entities disappear. If the button is clicked again, the *SceneViewer* will not be launched normally and the application will only enter the full-screen mode instead of AR mode. We observed 13 such issues in our dataset.

Incompatible Operating Systems (OS): When using devices with iOS 10 to access the VR mode of the applications developed using *A-Frame*, the whole scene becomes misty and it is hard for users to recognize the objects (issue #2008 [45]). As shown in Figure 6, compared to the right image displayed on Android devices, the stone and plants in the left image appear to be blurry. We found 11 such issues in our dataset.

As the ecosystem of WebXR is rapidly developing, new devices and software are emerging and evolving actively.

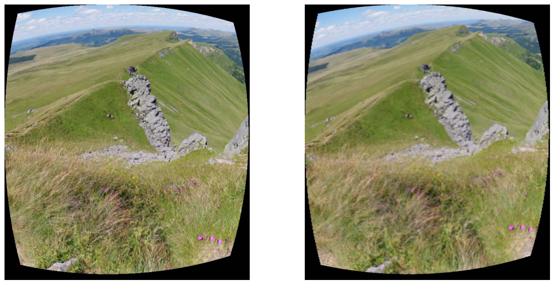

(a) Normal (b) Misty

Fig. 6: Incompatible OS causes the whole scene to be misty.

Consequently, developers may not be familiar with, or even aware of, the rules, usages, and caveats of certain hardware and software due to their rapid evolutions. This in turn results in prevalent compatibility issues in WebXR applications.

Finding 5: *Runtime incompatibility is the most common root cause of WebXR bugs. Around 21% of our studied bugs occurred due to this reason.*

B. Event Handling Mistakes (EVENT)

WebXR applications are highly interactive. A typical WebXR application may need to deal with three kinds of events: (1) lifecycle events, (2) system events, and (3) custom events. While the sophisticated event-handling mechanisms in WebXR projects enable flexible message exchanges between modules and versatile user interactions, they also raise the bar for developers, who need to be extremely familiar with these mechanisms in order to handle the event logic correctly. Unfortunately, in practice, developers of WebXR projects often make event handling mistakes. As shown in Figure 1(b), 76 (20.65%) of our studied WebXR bugs are caused by such mistakes. We present some examples in the following.

Mishandling of Lifecycle Events (LIFECYCLE): Lifecycle events arise when a component in a WebXR application goes through different life stages, such as initialization, update, destroy, play, suspension, and so on. Mishandling of such events may lead to various consequences. Take *playcanvas engine* as an example. The project developers did not *"correctly destroy the skybox model when updating skybox properties in the scene"*, which caused memory leak according to the issue report (issue #1554 [48]). In our dataset, 56 (15.22%) bugs are caused by mistakes in handling lifecycle events.

Mishandling of System Events (SYSTEM): System events in WebXR applications concern multiple application components and arise when the status of the system changes. For instance, an old version *Babylon.js* dispatches camera-change events in a wrong way (issue #5881 [49]), and the applications depending on *Babylon.js* behaved incorrectly due to this mistake: if users click on the *Enter VR* button, the "active camera change" message will show up over and over again instead of appearing only once. Bugs caused by mistakes in handling system events are also common and we observed 17 cases in our dataset.

Listing 2: A bug caused by mishandling of custom events

```
update: function (oldData) {
 // All sound values set. Load in `src`.
 if (srcChanged) {
   var self = this;
   this.loaded = false;
   this.audioLoader.load(data.src, function (buffer) {
     self.loaded = true;
     // Remove this key from cache, otherwise we
         can't play it again
     if (self.data.autoplay || self.mustPlay) {
         self.playSound(); }
+     self.el.emit('sound-loaded');
   });
 }
},
```

Mishandling of Custom Events (CUSTOM): Custom events are application-specific events that are only emitted and listened by certain application components. Although bugs caused by mistakes in handling custom events are rare (we only observed two cases in our dataset), diagnosing and fixing such bugs may not be an easy task. For example, *A-Frame* suffered from a bug that would make applications based on the framework keep playing the last sound even after the source changes. This bug occurred because developers forgot to emit the custom event *sound-loaded* when the source changes. It was fixed by adding an event emission statement as shown in Listing 2 (pull request #2457 [50]).

Listing 3: A bug caused by DUI

```
onEnterVR: function () { // Save pose.
+  if (!this.el.sceneEl.checkHeadsetConnected()) {
+    return;
+  }
  this.saveCameraPose();
  this.el.object3D.position.set(0, 0, 0);
  this.el.object3D.updateMatrix();
},
onExitVR: function () { // Restore the pose.
+  if (!this.el.sceneEl.checkHeadsetConnected()) {
+    return;
+  }
  this.restoreCameraPose();
  this.previousHMDPosition.set(0, 0, 0);
},
```

Finding 6: *Event handling mistakes are the second most common category of root causes of WebXR bugs, possibly because WebXR applications often have a large number of functional components and modules that need to communicate with each other via events.*

C. Improper Handling of Diversified User Interactions (DUI)

The interaction space of WebXR applications is huge. Users could interact with a WebXR application in many different ways using various devices. Developers should program their WebXR applications to deal with user interactions correctly. However, it is difficult for developers to anticipate and handle all possible interaction scenarios, especially abnormal ones. We observed that developers often failed to properly handle diversified user interactions. There are 36 bugs in our dataset arose due to such mistakes. For example, *A-Frame* does not check whether the headset is connected when users try to enter the VR mode in desktop 2D or non-VR situation (issue

#3902 [45]). Such negligence of checking users' interaction status caused rendering issues in the applications developed based on *A-Frame*. As shown in Listing 3, the bug was fixed by adding code to check whether the headset is connected in respective situations.

Finding 7: *WebXR applications are highly interactive. Developers often fail to anticipate and handle all possible user interactions, resulting in many bugs.*

D. Wrong Arguments or Configurations (ARG)

In complex software, different modules may be configured separately and rely on argument passing for communication. WebXR applications involve many modules and improper argument usages or configurations can cause bugs. We observed 27 such bugs in our dataset, whose root causes can be further divided into the following three categories:

Misconfiguration (MISCONFIG): 16 bugs in our dataset were caused by the wrong configuration of the interfaces exposed to external browsers and libraries. For example, the *z-index* property of CSS is used to configure the elements' stack order, indicating which element is on top of the others [51]. Wrong settings of *z-index* in *A-Frame* caused an invalid cascade of the UI components, making the ones in the back invisible in *A-Frame*'s dependents (issue #3389 [45]).

Argument Misuse (ARGMISUSE): Seven bugs in our dataset were caused by argument misuses. For example, in *A-Frame*'s dependents, the cursor appears in the wrong place rather than in the center of the scene (issue #3841 [45]). This bug occurred because a wrong argument was passed to `THREE.WebGLRenderer.vr.setPoseTarget()`, a function in the *Three.js* library [47]. Listing 4 shows the fix.

Listing 4: An example of argument misuse

```
setupRenderer: {
  value: function(){
-   renderer.vr.setPoseTarget(this.camera);
+   if (this.camera) {
+    renderer.vr.setPoseTarget(this.camera.el.object3D);
+   }
    this.addEventListener('camera-set-active', () => {
-    renderer.vr.setPoseTarget(self.camera);
+    renderer.vr.setPoseTarget(self.camera.el.object3D);
    });
  },},
```

Argument Missing (ARGMISS): Four bugs in our dataset were caused by the absence of certain necessary arguments. For instance, the application *mozilla/hubs* missed the *touch-cancel* argument for touch events, which led to *"menu does not open after virtual joysticks are activated"*, as reported by users (issue #289 [44]).

Finding 8: *Bugs caused by misconfigurations or wrong arguments are frequently observed, possibly because developers need to deal with the diverse interfaces between the many modules in WebXR architecture.*

E. Buggy Dependencies (DEPEND)

For 15 bugs in our dataset, the developers of the projects, to which the bugs were reported, explicitly stated in the issue reports that the bugs were caused by their project's dependencies. For example, issue #3623 [45] of *A-Frame* described the problem that an individual audio resource in an *A-Frame* based application could not be reused by more than one entity. In particular, if the audio resource had already been loaded in one entity, developers would fail to load the same resource in other entities. As pointed out by *A-Frame* developers, this bug was caused by the *AudioLoader* problems of *Three.js*, an important dependency of *A-Frame*.

Finding 9: *In practice, WebXR projects may be built on top of various other projects. Buggy dependencies are also threats to the reliability of WebXR applications.*

F. Redundant Operations (RDDOP)

Lastly, unnecessary or duplicate operations can also cause bugs. We observed 11 such cases in our dataset. For instance, in the *playcanvas engine* project, disabled script components are iterated twice for every frame in both the update and postUpdate processes (issue #1265 [48]). Such repetitions are apparently unnecessary, which would waste computational resources and lead to performance degradation.

Finding 10: *Redundant operations usually result in performance issues.*

VI. RQ3: Uniqueness

This section presents the comparison between WebXR bugs and bugs in JavaScript programs and web applications.

A. Comparison with Bugs in JavaScript Programs

We compared WebXR bugs with bugs in other JavaScript programs, including both server-side and client-side ones.

Server-Side JavaScript Programs: We chose BugsJS [38], which is a benchmark of JavaScript bugs collected from popular server-side programs, for the comparative study. Specifically, BugsJS contains 453 real bugs, each of which has been manually validated. For comparison, we used our taxonomy to classify bugs in BugsJS by investigating the issue report, category, and fixing strategy information of each bug. We made three observations from the comparisons:

- For symptoms, since bugs in BugsJS are found in UI-less server-side JavaScript programs based on *Node.js*, they do not cause rendering, low FPS or resolution problems.
- For root causes, we did not find any bugs in BugsJS that are caused by incompatible runtime environment or improper handling of diversified interactions, since server-side JavaScript programs do not require complicated user interactions with special devices and do not need to be adapted to different software/hardware.
- For those types of bugs that appear in both BugsJS and WebXR projects, their proportion diverges a lot. First, only 0.66% (3/453) bugs in BugsJS are caused by misconfiguration (more precisely, incorrect API config) while the proportion is 4.35% (16/368) in our WebXR dataset. Second, 33.33% (151/453) bugs in BugsJS are caused by argument

misuse, however, only 1.90% (7/368) of our studied WebXR bugs share the same root cause.

Client-Side JavaScript Programs: Ocariza et al. [39] performed an empirical study on 317 JavaScript bugs at the client side (to ease presentation, we call these bugs "client JS bugs"). They reported the characteristics of the bugs in detail, including fault categories, consequences, causes, browser specificity of bugs, etc. Their reported fault categories and cause patterns are similar to the root causes summarized in our work. Therefore, we directly compared their bug taxonomy with ours and made the following observations.

- There are plenty of client JS bugs caused by improper handling of user interactions. However, such bugs are mostly related to incorrect validation of user inputs and have nothing to do with special user interaction devices. This is obviously different from WebXR bugs, many of which arise from diversified user interactions via special XR devices.
- Different browsers have diverse JavaScript engines, which may interpret JavaScript code differently. About 9% of the client bugs are caused by incompatible browsers. Similarly, we observed that WebXR bugs can also be caused by browsers and such bugs account for 3.53% of our studied WebXR bugs. However, compared to traditional JavaScript programs, the runtime environment of WebXR applications is more complicated. Incompatible XR devices and operating systems are common root causes of WebXR bugs. Such root causes are not observed in Ocariza et al.'s study.
- The majority of client JS bugs are caused by *incorrect method parameters* (74%). A similar root cause, i.e., *wrong arguments or configurations*, is also observed in our study. However, bugs caused by wrong arguments only account for 7.34% in our dataset. The majority of our studied WebXR bugs are caused by incompatible runtime environment, event handling mistakes, and improper handling of diversified user interactions. We conjecture that this huge difference might be attributed to two reasons. First, WebXR device API is a new standard. Browsers and various XR devices do not fully support the standard yet. Second, developers are not familiar with WebXR programming and thus make various mistakes in handling user interactions and events.

As for bug symptoms, Ocariza et al. divided the bug consequences (symptoms) roughly into two categories: *code-terminating* and *output-related*. Similar bug symptoms are also observed in our dataset. However, since their bug dataset is not available, we cannot perform more detailed comparisons.

B. Comparison with Bugs in Web Applications

Marchetto et al. [40] studied 676 bugs in general web applications. They proposed a three-level taxonomy by analyzing the root causes and symptoms of the bugs. We compared our findings with theirs and made the following observations, which are essentially similar to the observations presented in Section VI-A:

- Similar to WebXR bugs, there are web application bugs caused by incompatible browsers. However, no bugs in general web applications are reported to be caused by incompatible operating systems or device models.
- Only four of the 676 bugs are caused by event handling mistakes. Comparatively, in our WebXR bug dataset, the proportion of such bugs is up to 20.65%. In addition, Marchetto et al.'s work did not report bugs caused by improper handling of diversified user interactions, yet bugs with this root cause are common in our dataset.

As discussed above, the differences are likely attributed to the immaturity of the WebXR ecosystem and developers' inexperience of WebXR programming.

Finding 11: *WebXR bugs significantly differ from bugs in conventional JavaScript programs and web applications in terms of both symptoms and root causes. One prominent difference is that many WebXR bugs arise from incompatible devices and improper handling of diversified user interactions. Such bugs are not found in conventional JavaScript programs or web applications.*

VII. Discussions

A. Threats to Validity

Internal Validity. The major threat to internal validity is that our manual analysis process could be error-prone. We understand that the results of our manual inspection and labeling could be biased since previous domain knowledge and experience might influence the human evaluators' decisions. In order to reduce the threat, we adopted the open coding approach [42]. Three co-authors followed a rigorous process to inspect the bugs separately and then cross-checked the results and reached consensus after seven rounds of discussions.

External Validity. A threat to the external validity is the comprehensiveness and representativeness of the WebXR bugs we collected, which could influence the generality of our findings. To mitigate the threat, we first collected 33 top-ranked WebXR projects on GitHub. We then searched both the issue tracking systems and the release notes of these projects to collect bugs. We also released our dataset for public scrutiny. Another threat is that we only compared our work with several existing pieces of work since it is hard to compare empirical findings of different studies, which often adopt different analysis strategies. In future, we will further study the uniqueness of WebXR bugs by comparing with more related studies.

B. Implications of Our Findings

Our findings on bug symptoms reveal that WebXR applications often suffer from abnormal interactions and unexpected crashes, which could significantly degrade user experiences. However, testing technologies targeting at conventional software may not be adequate for testing WebXR applications due to the specific features of these applications. To begin with, the wide-spread fragmentation problem in the WebXR ecosystem brings an obvious challenge to testing WebXR applications. Its induced runtime incompatibility is the most common root cause of WebXR bugs (Finding 5). An effective WebXR testing framework should be adaptive to a variety of browsers, WebXR frameworks, device models, and operating systems when handling diverse user interactions via various devices. Next, a large percentage of WebXR bugs are application-specific (e.g., 44.02% functional issues and 23.10% rendering issues according to Findings 1 and 2), making it difficult to construct general test oracles. Furthermore, we found that the interaction mechanisms and input types are diversified (Finding 7). As the search space of WebXR applications' inputs is huge (e.g., the rotation degree in the three-dimension space for both the controllers and the headset), achieving satisfactory test coverage seems to be difficult. We believe that our study could help WebXR developers make better-informed decisions on developing and testing their WebXR projects. Our findings can also guide future research on this topic.

VIII. Related Work

A. Empirical Studies of Bugs in Traditional Software

JavaScript Bugs. JavaScript has been widely used for both client-side and server-side programming. Many researchers have studied the characteristics of bugs in JavaScript programs. Ocariza et al. [39] conducted a thorough study of bugs in client-side JavaScript programs. They analyzed and categorized the bugs they collected from multiple aspects, including fault types, consequences, causes, impacts, etc. As for consequences, they roughly classified them into two classes, namely, code-terminating and output-related. In comparison, our classification of symptoms is more detailed. We classified the symptoms of WebXR bugs into nine categories and paid attention to both functional and non-functional issues. Gyimesi et al. [38] presented a benchmark named BugsJS, which contains 453 server-side JavaScript bugs. They also analyzed the characteristics of the bugs in BugsJS. Specifically, they looked at the bug details at the code level and studied why did the bugs appear, where did the faults lie in the source code, and how did developers fix the bugs. Our work classified root causes of WebXR bugs from a higher-level perspective, such as which parts developers didn't handle well or design correctly. There are also other studies on JavaScript bug patterns [52]–[54], which are earlier than Gyimeisi et al's study [38], and studies on specific topics like code smells [55], [56], security [57], and performance [58] issues of JavaScript. Due to the page limit, we do not further discuss them in detail.

Bugs in Web Applications. Web applications are an important type of software. There is a large body of research on the defects of general web applications. Marchetto et al. [40], [59] built a thorough bug taxonomy of web applications. They constructed an initial taxonomy based on domain knowledge of web applications' properties, and then iterated multiple times to analyze and classify each bug and refined taxonomy according to the characteristics of real-world bugs. Ma et al. [60] applied an orthogonal defect classification framework to analyze and classify web defects according to several specific fields of server log information. Guo et al. [61] analyzed real-world fault data and proposed a representative classification of web application bugs. In our work, we studied the characteristics of bugs in WebXR applications, which can be viewed as a special type of web applications. We compared our findings with those reported in Marchetto et al.'s work [40]. We found that although WebXR bugs are similar to web application bugs in some aspects (e.g., both can be caused by incompatible browsers), they have their own uniqueness and are worth further studies in the future (see Section VI-B).

B. Testing for Web Applications

In the past decade, researchers have proposed various techniques to test web applications. For example, fuzzing [62]–[64] is a widely-used web testing method, which randomly generates inputs and keeps monitoring the executing status of web applications to detect unexpected behaviors. Considering that random fuzzing may not be effective in some cases, existing work also proposed improved strategies such as guided fuzzing and stateful fuzzing to detect bugs in web applications [65]. Besides fuzzing, model-based approaches [66]–[68] were also proposed to test web applications. For instance, Mesbah et al. [66] designed an automatic testing approach, named Atusa, to test Ajax-based web applications based on a pre-inferred state-flow graph. Such states involve all *client-side UI states*. Atusa leverages genetic algorithms to extract paths and generate inputs on the pre-inferred model. In recent years, with the advances of reinforcement learning (RL) algorithms, researchers also proposed to leverage RL to test web applications. For example, Liu et al. [69] proposed a method to guide agents to discover suitable actions and execute tasks in web scenarios, where action space is huge and *rewards* for agents are rare.

As discussed in Section VII-B, testing WebXR applications faces unique challenges. Whether the above testing methods could help with WebXR testing needs further investigation.

IX. Conclusion

With the rapid development of network infrastructure (5G) and XR technologies, WebXR may become an important human-computer interaction mechanism in the next decades. In this paper, we conducted the first empirical study of WebXR bugs. We collected a dataset containing 368 bugs from 33 open-source WebXR projects. We studied the symptoms and root causes of the bugs and built the bug taxonomy from the two aspects. To understand the uniqueness of WebXR bugs, we also compared them with bugs in conventional JavaScript programs and web applications. We hope that our empirical findings can facilitate future research and provide useful guidance to practitioners. Our dataset can be found at both https://sites.google.com/view/webxr-bug-study/ and Zenodo (https://doi.org/10.5281/zenodo.3992105).

Acknowledgments

This work is supported by the National Key Research and Development Program of China (Grant No. SQ2019YFE010068), the National Natural Science Foundation of China (Grants #61932021 and #61802164), and the Australian Research Council (Grant No. DP200101328).

References

[1] "Wikipedia of virtual reality," https://en.wikipedia.org/wiki/Virtual_reality, 2020.

[2] "Wikipedia of augmented reality," https://en.wikipedia.org/wiki/Augmented_reality, 2020.

[3] "What is mixed reality," https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality, 2020.

[4] "Oculus VR Games and Apps," https://www.oculus.com/experiences/rift/, 2020.

[5] "Virtual Reality Pain Reduction – Human Photonics Laboratory," http://www.vrpain.com/, 2020.

[6] "VirTra," https://www.virtra.com, 2020.

[7] S. Thompson, "VR Applications: 21 Industries already using Virtual Reality," https://virtualspeech.com/blog/vr-applications, 2020.

[8] J. Marchant, "Surgeon gives patients VR," http://www.bbc.com/future/story/20170202-the-surgeon-using-virtual-reality-instead-of-sedatives, 2020.

[9] "WebXR Device API," https://www.w3.org/TR/webxr/, 2020.

[10] "Immersive Web Developer Home," https://immersiveweb.dev/, 2020.

[11] "WebGL Overview," https://www.khronos.org/webgl/, 2020.

[12] "Mozilla - WebXR Device API," https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API, 2020.

[13] J. Rossi, "Bring Immersive Web Content into VR," https://developers.facebook.com/blog/post/v2/2019/04/30/immersive-web-content-vr/, 2020.

[14] C. T. Duque, "These Next-Gen Interactive Content Formats Are Hungry For A Critical LMS Mass: WebXR, 3D, Maps and More," https://www.lmspulse.com/2019/content-formats/, 2020.

[15] "npm - Build amazing things, take your JavaScript development up a notch," https://www.npmjs.com/, 2020.

[16] "GitHub - the world's leading software development platform," https://github.com, 2020.

[17] "GitHub Repository of aframevr/aframe," https://github.com/aframevr/aframe, 2020.

[18] "Wikipedia of WebVR," https://en.wikipedia.org/wiki/WebVR, 2020.

[19] "WebVR - Bringing Virtual Reality to the Web," https://webvr.info/, 2020.

[20] A. Vrignaud and L. Bergstrom, "Bringing Mixed Reality to the Web," https://blog.mozilla.org/blog/2017/10/20/bringing-mixed-reality-web/, 2020.

[21] "W3C First Public Working Draft, 5 February 2019," https://www.w3.org/TR/2019/WD-webxr-20190205/, 2020.

[22] "WebVR API - Mozilla MDN web docs," https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API, 2020.

[23] "W3C Working Draft, 21 May 2019," https://www.w3.org/TR/2019/WD-webxr-20190521/, 2020.

[24] "W3C Working Draft, 10 October 2019," https://www.w3.org/TR/2019/WD-webxr-20191010/, 2020.

[25] "W3C Working Draft, 24 July 2020," https://www.w3.org/TR/2020/WD-webxr-20200724/, 2020.

[26] "WebXR Device API - Editor's Draft," https://immersive-web.github.io/webxr/, 2020.

[27] "GitHub Repository of AR.js," https://github.com/jeromeetienne/AR.js, 2020.

[28] "GitHub Repository of BabylonJS/Babylon.js," https://github.com/BabylonJS/Babylon.js, 2020.

[29] "GitHub Repository of playcanvas/engine," https://github.com/playcanvas/engine, 2020.

[30] "GitHub Repository of google/model-viewer," https://github.com/google/model-viewer, 2020.

[31] "GitHub Repository of donmccurdy/aframe-extras," https://github.com/donmccurdy/aframe-extras, 2020.

[32] "GitHub Repository of mozilla/hubs," https://github.com/mozilla/hubs, 2020.

[33] "GitHub Repository of donmccurdy/aframe-physics-system," https://github.com/donmccurdy/aframe-physics-system, 2020.

[34] "GitHub Repository of mozilla-mobile/webxr-ios," https://github.com/mozilla-mobile/webxr-ios, 2020.

[35] "GitHub Repository of immersive-web/webxr-polyfill," https://github.com/immersive-web/webxr-polyfill, 2020.

[36] "GitHub Repository of supermedium/moonrider," https://github.com/supermedium/moonrider, 2020.

[37] "GitHub Repository of engaging-computing/MYR," https://github.com/engaging-computing/MYR, 2020.

[38] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Á. Beszédes, R. Ferenc, and A. Mesbah, "Bugsjs: a benchmark of javascript bugs," in *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*. IEEE, 2019, pp. 90–101. [Online]. Available: https://doi.org/10.1109/ICST.2019.00019

[39] F. S. O. Jr., K. Bajaj, K. Pattabiraman, and A. Mesbah, "An empirical study of client-side javascript bugs," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, 2013, pp. 55–64. [Online]. Available: https://doi.org/10.1109/ESEM.2013.18

[40] A. Marchetto, F. Ricca, and P. Tonella, "An empirical validation of a web fault taxonomy and its usage for web testing," *J. Web Eng.*, vol. 8, no. 4, p. 316–345, Dec. 2009.

[41] "WebXR emulator extension - a browser extension which helps to run and test WebXR content." https://github.com/MozillaReality/WebXR-emulator-extension, 2020.

[42] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.

[43] "Issue tracker of supermedium/moonrider on GitHub," https://github.com/supermedium/moonrider/issues, 2020.

[44] "Issue tracker of mozilla/hubs on GitHub," https://github.com/mozilla/hubs/issues, 2020.

[45] "Issue tracker of aframevr/aframe on GitHub," https://github.com/aframevr/aframe/issues, 2020.

[46] "Issue tracker of google/model-viewer on GitHub," https://github.com/google/model-viewer/issues, 2020.

[47] "Three.js – JavaScript 3D library," https://threejs.org/, 2020.

[48] "Issue tracker of playcanvas/engine on GitHub," https://github.com/playcanvas/engine/issues, 2020.

[49] "Issue tracker of BabylonJS/Babylon.js on GitHub," https://github.com/BabylonJS/Babylon.js/issues/5881, 2020.

[50] "Pull request tracker of aframevr/aframe on GitHub," https://github.com/aframevr/aframe/pulls, 2020.

[51] "CSS z-index Property," https://www.w3schools.com/cssref/pr_pos_z-index.asp, 2020.

[52] Q. Hanam, F. S. D. M. Brito, and A. Mesbah, "Discovering bug patterns in javascript," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, T. Zimmermann, J. Cleland-Huang, and Z. Su, Eds. ACM, 2016, pp. 144–156. [Online]. Available: https://doi.org/10.1145/2950290.2950308

[53] F. S. O. Jr., K. Pattabiraman, and B. G. Zorn, "Javascript errors in the wild: An empirical study," in *IEEE 22nd International Symposium on Software Reliability Engineering, ISSRE 2011, Hiroshima, Japan, November 29 - December 2, 2011*, T. Dohi and B. Cukic, Eds. IEEE Computer Society, 2011, pp. 100–109. [Online]. Available: https://doi.org/10.1109/ISSRE.2011.28

[54] F. S. O. Jr., K. Bajaj, K. Pattabiraman, and A. Mesbah, "A study of causes and consequences of client-side javascript bugs," *IEEE Trans. Software Eng.*, vol. 43, no. 2, pp. 128–144, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2016.2586066

[55] D. Johannes, F. Khomh, and G. Antoniol, "A large-scale empirical study of code smells in javascript projects," *Software Quality Journal*, vol. 27, no. 3, pp. 1271–1314, 2019. [Online]. Available: https://doi.org/10.1007/s11219-019-09442-9

[56] A. Saboury, P. Musavi, F. Khomh, and G. Antoniol, "An empirical study of code smells in javascript projects," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE Computer Society, 2017, pp. 294–305. [Online]. Available: https://doi.org/10.1109/SANER.2017.7884630

[57] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 270–283. [Online]. Available: https://doi.org/10.1145/1866307.1866339

[58] M. Selakovic and M. Pradel, "Performance issues and optimizations in javascript: an empirical study," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016,*

Austin, TX, USA, May 14-22, 2016, L. K. Dillon, W. Visser, and L. Williams, Eds. ACM, 2016, pp. 61–72. [Online]. Available: https://doi.org/10.1145/2884781.2884829

[59] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *Proceedings of the 9th IEEE International Symposium on Web Systems Evolution, WSE 2009, 5-6 October 2007, Paris, France*, S. Huang and M. D. Penta, Eds. IEEE Computer Society, 2007, pp. 31–38. [Online]. Available: https://doi.org/10.1109/WSE.2007.4380241

[60] L. Ma and J. Tian, "Web error classification and analysis for reliability improvement," *J. Syst. Softw.*, vol. 80, no. 6, pp. 795–804, 2007. [Online]. Available: https://doi.org/10.1016/j.jss.2006.10.017

[61] Y. Guo and S. Sampath, "Web application fault classification - an exploratory study," in *Proceedings of the Second International Symposium on Empirical Software Engineering and Measurement, ESEM 2008, October 9-10, 2008, Kaiserslautern, Germany*, H. D. Rombach, S. G. Elbaum, and J. Münch, Eds. ACM, 2008, pp. 303–305. [Online]. Available: https://doi.org/10.1145/1414004.1414060

[62] R. Hammersland and E. Snekkenes, "Fuzz testing of web applications," 2008.

[63] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: A state-aware black-box web vulnerability scanner," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, T. Kohno, Ed. USENIX Association, 2012, pp. 523–538. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/doupe

[64] F. Duchene, S. Rawat, J. Richier, and R. Groz, "Kameleonfuzz: evolutionary fuzzing for black-box XSS detection," in *Fourth ACM Conference on Data and Application Security and Privacy, CODASPY'14, San Antonio, TX, USA - March 03 - 05, 2014*, E. Bertino, R. S. Sandhu, and J. Park, Eds. ACM, 2014, pp. 37–48. [Online]. Available: https://doi.org/10.1145/2557547.2557550

[65] S. McAllister, E. Kirda, and C. Kruegel, "Leveraging user interactions for in-depth testing of web applications," in *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., vol. 5230. Springer, 2008, pp. 191–210. [Online]. Available: https://doi.org/10.1007/978-3-540-87403-4_11

[66] A. Mesbah, A. van Deursen, and D. Roest, "Invariant-based automatic testing of modern web applications," *IEEE Trans. Software Eng.*, vol. 38, no. 1, pp. 35–53, 2012. [Online]. Available: https://doi.org/10.1109/TSE.2011.28

[67] B. Yu, L. Ma, and C. Zhang, "Incremental web application testing using page object," in *Third IEEE Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015, Washington, DC, USA, November 12-13, 2015*. IEEE Computer Society, 2015, pp. 1–6. [Online]. Available: https://doi.org/10.1109/HotWeb.2015.14

[68] M. Biagiola, A. Stocco, F. Ricca, and P. Tonella, "Diversity-based web test generation," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, pp. 142–153. [Online]. Available: https://doi.org/10.1145/3338906.3338970

[69] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=ryTp3f-0-