An Empirical Study of Testing Practices in Virtual Reality Software

Isabella Attisano[1], Xue Qin[1], Dhia Elhaq Rzig[2], Foyzul Hassan[2]
[1]Villanova University, [2]University of Michigan-Dearborn

Abstract

Software Testing is the critical step in software development to safeguard against problems such as significant financial losses, security breaches, and performance issues, before the software release. Many research approaches have been developed to improve the accuracy and efficiency of Software Testing. Among all types of software, Virtual Reality(VR), a form of software that simulates a 3D world paralleling true reality, has grown popular [1] recently due to its significant impact on the user's senses and physical wellbeing. However, little research effort [2] has been paid on software testing of VR. Moreover, VR software is difficult to test due to its different features compared to normal software, such as dependence on user interaction and special VR frameworks.
This empirical study aimed to conduct a manual analysis to identify the testing practices of existing VR projects. Specifically, we focus on finding the unit test cases among all the VR projects collected from GitHub. Furthermore, we endeavored to discover other types of testing such as system testing, integration testing, and performance testing. In the future, we will use the study result to identify the primary limitations of VR testing and suggest comprehensive improvements to increase the effectiveness of VR software testing.

Figure 1: An example framework for VR software

Background

Virtual Reality (VR) is a form of computer simulation that allows a user to interact with their surroundings as if they were in the real world by simulating the five senses in order. In addition to VR technology, other technological realities include augmented Reality (AR), Mixed Reality (MR), and extended reality (XR). Figure 1 shows an example framework for developing virtual reality software and how virtual reality relies on both software and hardware.

In software testing, unit testing is crucial as it allows a tester to better understand specific code performance by testing small, single units of projects such as a single method. Unity platform is an Integrated Development Environment (IDE) that allows developers to create 3D worlds by employing VR devices such as Oculus or HoloLens and creating and managing objects and scenes. . Unity includes the Test Runner framework to execute test cases for VR projects and Unity also integrates the NUnit library to support the general test cases in C#.

Objectives

1. Conduct a manual analysis of a collection of VR projects from Github to discover the quantity of existing testing practices.
2. Among all test practices, uncover frequent used test granularity and type.

Methodology

Study Setup

- A total of 98 VR GitHub repositories were re-collected from an existing study [3]. All the projects were updated to the newest version by the time we conducted the study.
- These repositories were selected based on the criteria: #C-based VR projects, Unity platform supported, and contained at least 100 commits.

Objective 1 Analysis

- To discover the quantity of existing testing practices among all project repositories, we created a static analysis script which automatically report all the locations in project source codes that has "test" keyword. These locations include variables, statements, methods, classes and annotations.
- Then, we carried out a manual analysis of all the projects to validate the automated discovered test methods. We excluded all the wrongly or repeated labelled test methods, and then located and recounted any unreported test methods.

Objective 2 Analysis

- As shown in Figure 2, different types of test approaches have been adopted to evaluate the quality and assurance of the software. To uncover frequent used test granularity and type, we manually classify each test method and categorized them to one of the testing types (e.g., system testing, performance testing, and unit testing) by reading and understanding the source code.

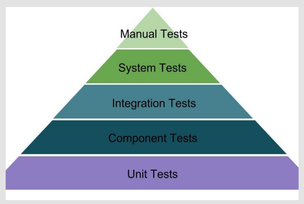

Figure 2: Software Testing Types

```
public class PacmanScoreTests
{
    private PacmanScore pacmanScore;

    [SetUp]
    public void Setup()
    {
        pacmanScore = MonoBehaviour.Instantiate(Resources.Load<GameObj
        pacmanScore.Start();
    }

    [TearDown]
    public void Teardown()
    {
        Object.Destroy(pacmanScore.gameObject);
    }

    [Test]
    public void AddingScoreWorks()
    {
        pacmanScore.AddScore(50);
        Assert.AreEqual(50, pacmanScore.GetScore());
    }
}
```

Figure 3: VR Testing Example 1

- Figure 3 shows one example of the Unit Test in Virtual Reality Project. It includes the NUnit annotations: `[Setup]`, `[TearDown]`, `[Test]`. Besides this general format, Unity platform also provides annotation such as `[UnityTest]` to support the Unit Test using Unity framework.

Methodology (cont.)

- If the program in its entirety was tested, we then classified this as a system test. System tests could typically be identified if the class was identified to be a test class and contained methods such as `Start()`, `Update()`, and `OnDestroy()`. This indicated it was testing an entire system rather than a single unit or a group of units. Figure 4 shows an example of the system testing in project MixedRealityToolkit-Unity.

```
        /// </summary>
        public void RunTest()
        {
            StartCoroutine(TestAnchors());
        }


        private void Start()
        {
            UpdateUI();
        }

        private void UpdateUI()
        {
            if (text != null)
            {
                text.text = "Anchor: " + grid.Anchor;
            }
            grid.UpdateCollection();
        }
```

Figure 4: VR Testing Example 2

- Unity platform also provides its own featured testing framework that come with the methods like `TestStarted()`, `TestFinished()`, `RunStarted()`, and `RunFinished()`.
- Lastly, performance tests were identified if the method appeared to test attributes of the project such as responsiveness, speed, etc., rather than units like methods.

Study Results

Objective 1 Results

- Among 98 VR projects, only 61.2% (60 out of 98) projects include one or more test methods.
- The average number of tests per project is 53 with a standard deviation of 129.6, where the maximum number of test methods is 896 and the minimum is 2.

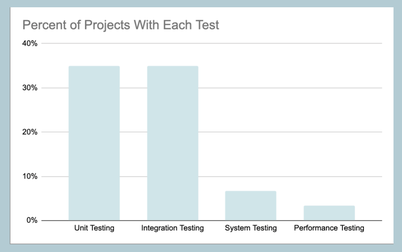

Figure 5: Testing Types Percentage

Objective 2 Results

- Among the 180 test methods we identified in 98 VR projects, the percentage for each test type is shown in Figure 5, where Unit Testing holds 35%, Integration Testing holds 35%, System Testing holds 6.67%, and Performance Testing holds 3.3%.

Results Discussion

Unit Testing Practices

- Only 35% of the studied projects contained unit testing. The average percentage of unit tests per project was 38%, and after removing the outliers, it gives us roughly 29%.

Invalid Test Methods in VR Packages

- Among all the 60 projects that includes the test methods, 30.7% of them contained test methods solely from existing VR packages. For instance, 52.6% of them are using StreamVR package.
- However, testing Methods in VR packages are predefined as examples, and later we found that they are not being executed when running the test tools in Unity.

Conclusion and Future Work

By studying and manually analyzing a series of 98 Virtual Reality projects, it became apparent that software testing of virtual reality projects is limited. Only 61.2% of the 98 projects contained at least 1 test method, and only 15% of those 60 contained more than 10 test methods. Among these 60 projects, 30.7% contained test methods from existing packages. This indicates that most developers do not write their test methods. This could stem from the difficulty of implementing VR tests due to VR's dependence on user input and interaction. Moreover, automatic test generation is difficult for VR projects, and consequently, most testing is done manually by developers. However, it is important to note that 35% of these 60 projects did contain unit testing.
Our future work in this area will seek to answer the following research questions: what is the design quality of test practices in VR software. By completing these objectives, we may enlighten the future researchers to find the potential solutions to improve the test efficiency of the VR Software.

References

[1] J. Molina, X. Qin, and X. Wang, "Automatic Extraction of Code Dependency in Virtual Reality Software." *IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. Web. doi:10.1109/ICPC52881.2021.00043.
[2] S. Andrade, F. Nunes and M. Delamaro, "Towards the Systematic Testing of Virtual Reality Programs," in *2019 21st Symposium on Virtual and Augmented Reality (SVR)*, Rio de Janeiro, Brazil, 2019 pp. 196-205. doi: 10.1109/SVR.2019.00044
[3] F. Nusrat, F. Hassan, H. Zhong and X. Wang, "How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects," *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 473-485, doi: 10.1109/ICSE43902.2021.00052.

Acknowledgements

- This work is a collaboration research with University of Michigan-Dearborn.
- This work is done by the support of Department of Computer Science, Villanova University
- Contact xue.qin@villanova.edu for more information