AN EMPIRICAL ANALYSIS OF UNITY PERFORMANCE BUGS

by

FARIHA NUSRAT, B.Sc.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTERS IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Xiaoyin Wang, Ph.D., Chair
Palden Lama, Ph.D.
Ali Tosun, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
July 2020

DEDICATION

I would like to dedicate this thesis to my parents, to my husband Foyzul and to rest of my family.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Dr. Xiaoyin Wang. Without his guidance, assistance and valuable time it would not be possible for me to finish my Master's thesis. Dr. Wang always helped me to develop an good understanding of the subject and gave me suggestions to improve the quality of my research results. Besides my supervisor, I would like to thank the rest of my dissertation committee: Prof. Palden Lama and Prof. Ali Tosun for their support, constructive feedback, and challenging questions.

I thank the staff members of Computer Science department, Susan Allen and John Meriwether for all the administrative work they do for us. They helped me a lot during my Masters at UTSA.

My sincerest gratitude and appreciation goes to my husband Foyzul Hassan for his constant support and assistance. I would also like to thank my parents and my brother for their never-ending love, care, support and encouragement throughout the completion of this Master of Science degree.

(Notice: If any part of the thesis/dissertation has been published before, the following two paragraphs should be included without alteration).

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Masters Thesis/Recital Document or Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.

July 2020

AN EMPIRICAL ANALYSIS OF UNITY PERFORMANCE BUGS

FARIHA NUSRAT, M.Sc.
The University of Texas at San Antonio, 2020

Supervising Professors: Xiaoyin Wang, Ph.D.

In the age of modern technology various game engines or game frameworks are used to develop games for different platforms and Unity is one of the most popular and commonly used game engines. Games, real-time 3d animations, etc. can be created using Unity's fully integrated development environment. Though Unity is mainly a game engine, it is also used to create various AR or VR apps used for medical, education and business purpose. Since Unity applications are being downloaded in a large number, it's high time to focus on the performance of Unity. For any kind of software product, performance is very important to indicate its overall quality. Software's quality or performance is deteriorated by different performance bugs and it can result in poor user experience, wasted computer resources. Though Unity is widely used and performance bug is as damaging as functional bugs, till now no research has been done on Unity's performance bugs. To provide researchers and developers further working on detecting, localizing and fixing performance bugs of Unity, we prepared a dataset of 230 performance bug fixing commits collected across 100 open source Unity project from GitHub. With our detail analysis we generated Unity's performance bug taxonomy which can be helpful for developers and future researchers. With our AST level code change analysis we identified that Unity's call-back method such as Update, Start, etc. are prone to performance issues and Unity performance fix commits are large due to code dependency issues. Apart from that our code change analysis also identified that performance fix can be in combination of source code and prefab files or sometimes only prefab files.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Unity is one of the most popular cross-platform game engines that is designed to support and develop 2D and 3D video games, simulations for computers, virtual reality, consoles and mobile devices platform [6]. This engine can also be used to make high quality games, design and develop 3D social network gaming platforms [11]. Unity is also used to create some VR/AR applications, that are used for various medical, education or business purposes. The main advantage of Unity 3D is that the editor runs on Windows and Mac OS X and can produce games for Windows, Mac, Web browsers, Wii, iOS (iPhone, iPod, Touch, and iPad), Android, Xbox 360, and Playstation 3 [16]. For game developers who want to start game development, Unity is an excellent and recommended platform [13]. Developers can save time and effort because of complete toolset, intuitive workspace and on the-fly play testing and editing feature of Unity [26]. Unity applications have been downloaded onto over 3 billion devices worldwide and applications made with the Unity engine have been downloaded over 24 billion times in the last 12 months [1]. Apart from Unity's all these advantages and popularity, researchers have found that Unity game platform perform slower than other engines [13]. And inspite of this, no research has been done regarding Unity's performance issue.

Among all the non-functional properties of programs, performance is the most important one [25, 45]. Software performance describes a software program's overall standard. Performance bug-programming error, inefficient source code can cause software quality or performance to deteriorate. It can result in poor user experience, deteriorated responsiveness, wasted computational resources [12, 35]. So, industry and research community have spent great effort to address performance bugs. But performance of modern game engines have never been formally analyzed [34]. That means, till now Unity's performance bugs or issues have not been addressed. Therefore, it is important to investigate which types of performance bugs mostly fixed in Unity to guide researchers and developers further working on detecting, localizing and fixing performance bugs of Unity.

For this purpose, first of all we downloaded 100 open source Unity projects from GitHub and created a dataset of 230 performance bug fixing commits from those projects. For choosing projects, we prioritized the projects with more than 100 commits, which is a indication that these are well-maintained projects. To find the performance commits we programmatically searched for some keywords in the commit messages and found 588 probable performance commits. To make sure if these commits are actually related to performance, we did manual analysis and finally got 230 actual performance commits.

With detailed root cause analysis, we identified Unity's performance bug fix taxonomy. That means, we categorized the bugs depending on their types and how they got fixed. This taxonomy can Be used for static analysis to find performance bugs and also generate automatic fix suggestions.

With AST level code change analysis, we provided some statistics regarding the types of methods that are prone to performance bugs, both with and without considering call-graph analysis and we found that Unity's call-back methods are prone to performance bugs. This gives an indication for developers that they should be more careful while writing Unity's call-back methods. We also tried to analyze the performance bug fix complexity in terms of Class, Method and Statement Change and found that performance Bugs Are complex in nature and require to change in multiple code locations. Finally to find the location of the bug fixes, we did the commit message analysis and identified that apart from source code related bugs, performance bugs can happen due to Unity GameObject Asset/Prefab Issues.

Summing up the above discussion, our study makes the following contributions:

- We prepared a dataset of Unity performance bugs that we used for our analysis and this data can also be used for future research.
- We generated Unity's performance bug taxonomy that can be used for static analyzer and automatic program repair techniques.
- Our analysis on performance bug identifies the nature or complexity of the bugs and gives

idea about the bug prone methods.

- We also found that for performance bug we need to fix in the asset/prefab files apart from source code.

The remaining part of this paper is organized as follows. After presenting a background related to *Unity* and performance analysis in Section 2, we describe our experiment methodology in Section 3. Section 4 presents the evaluation of our analysis, while Section 5 presents discussion. Related works and Conclusion will be discussed in Section 6 and Section 7, respectively.

CHAPTER 2: BACKGROUND

The purpose of this section is to provide the background of this study. Before going into details about my research work, I would like to discuss about some key terminology that we used in our work.

2.1 Unity

With ever-increasing demands of VR or AR application and Game Applications, developers are working with Unity Framework. Unity provides convenient integrated development environment through C# programming scripts. I would like to discuss about four basic terminology of Unity: **Gameobject**, **Asset**, **Prefab**, **Scripts**. The most important concept in the Unity editor is Gameobject. Gameobjects are the fundamental objects representing any kind of character, scenery, environment etc. But a gameobject cannot do anything on its own, it has no value without its properties. A gameobject acts as a container, to which you can add different components [7]. Any type of item that you can use in your game or program is an Asset. Assets can be created within Unity such as, animator controller, audio mixer etc. Assets may also come from a file that is created outside of Unity, such as, 3D model, audio file etc [4]. A gameobject can be created, configured and stored with all its components, properties, as well as with its child gameobject as a reusable asset and this is allowed by Unity's Prefab. That means, a prefab is a copy of a game object converted into a reusable asset New prefab instances can be created in the scene by using prefab asset as a template [8]. In Unity you can create your own components using Scripts. By using scripts, you can take assets in your scene and make them interactive. For scripts Unity supports the C# programming language [5].

2.2 Abstract Syntax Tree Based Code Differencing

To analyze and verify a program, code representation is a necessary step. An abstract syntax tree (AST) is used to represent the syntax of a programming language as a hierarchical tree-like struc-

ture. AST is one type of data structure that represents program structures to reason about syntax and semantics. All of the syntactical elements of the programming language are represented by AST. It focuses on the rules rather than elements that terminate statements. In AST each programming statement are broken down recursively into their parts and each node in the tree denotes a construct occurring in the programming language [27].

To study software evolution, differencing two versions of a program is the main pre-processing step. The evolved parts must be captured in an easy and understandable way. Text-based differencing tools are normally used by the developers, but the problem is that it does not provide a fine-grained representation of the change. That's why it is poorly suited for systematically analysing the changes. Recent algorithms have been proposed based on tree structures (such as the AST) to address this issue of code differencing, such as, ChangeDistiller and GumTree. These algorithms produce edit scripts that present detail operations to be performed on the nodes of a given AST to yield another AST corresponding to the new version of the code [27, 33].

2.3 Call Graph

A call graph represents calling relationships between subroutines/functions of a program. In a call graph, each node represents a procedure and every edge (a,b) indicates that procedure a calls procedure b. A call graph is a necessary prerequisite for most interprocedural analyses used in compilers, verification tools and program understanding tools [30]. There are two types of call graphs: dynamic, static. Dynamic call graph is a record of an execution of the program. It can be exact but it only describes that particular run of the program. On the other hand, static call graph represents every possible run of the program [44].

Call Graphs are widely used for Test Case Prioritization [32] and Test Case Selection [29]. For test case prioritization and selection, call graphs are used to obtain list of transitively relevant methods. Based on call graph analysis, test case with a higher number of invocation are prioritized or selected for regression. Apart from that, call graph is also used in change impact analysis [37]. Change in a single commit might impact or break whole software or package. To identify impact of

changes, one common approach is to perform change impact analysis. In change impact analysis call graph is also used to identify transitively relevant methods that can impacted due to recent changes.

CHAPTER 3: METHODOLOGY

To perform *Unity* performance bug analysis, we collected real-world bugs from GitHub projects. With the collected performance bug fix data, we performed i) Root cause analysis, ii) Code change-based AST diff analysis, and iii) Commit Analysis. Figure 3.1 shows overview of our overall methodology. In the subsequent sections, we discussed each of the components of our methodology.

3.1 Unity Performance Bug Data Collection

For Data Collection, we collected 100 Github projects with more than 100 commits. We used the commit threshold to ensure that the projects are well maintained and have sufficient fixed data. With these 100 projects, we programmatically searched for performance fix related keywords in commit message using Java Git library JGit [2]. As part of keyword search, we used **performance**, **speed up**, **accelerate**, **fast**, **slow**, **latency**, **contention**, **optimize**, and **efficient** keywords as search token. Prior research [15] on performance bug also used these keywords to find performance related bug fix. With this search strategy, we identified 588 probable performance-related issues. Later we manually checked the commits for the performance-related fix. We checked the commit messages thoroughly and also the fix for performance fix correctness. With our analysis, we identified 230 actual performance related bug fix. Since for Unity, we did not have any dataset for performance analysis, this dataset can contribute to further research on unity performance-related bugs.

3.2 Root Cause Analysis

With detailed root cause analysis, we identified Unity's performance bug fix taxonomy. That means, we categorized the bugs depending on their types and how they got fixed. We categorized these 230 bugs into 15 categories.

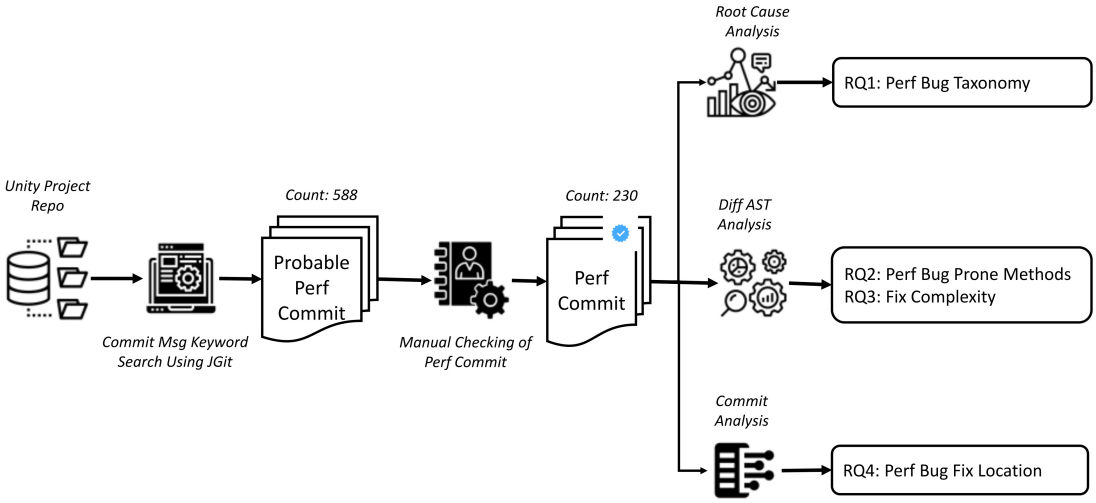

Figure 3.1: Overview of Unity Performance Bug Analysis

3.3 Code change-based AST diff analysis

For change/diff based analysis, we performed AST (Abstract Syntax Tree) level diff analysis. For C# AST tree generation, we did not find any opensource parser. So, we used srcML [3] a generic parsing research tool. With srcML, we converted C# code into XML format code. Then we loaded those XML format code and performed AST level diff with the state of the art diff tool GumTree [17]. With this approach, we can extract AST level code changes. Figure 3.2 shows the overview of C# AST level diff analysis.

Here is an example of the AST diff that we generated using our approach. We keep track of class name, method name, action such as insert, delete move etc. Also label and type. For example, here the class name is Answer, no function is used. And there is a single expression, whose type is "using name operator name", and label is "UnityEngine.UI". This "using" in C# is equivalent to Java "import". Also there is an "insert" action.

With AST level code changes, we analyzed which methods are prone to performance bug and the complexity of performance bug fix. We categorized performance bug fix methods as Unity Callback, Custom Callback, and User Defined Method during the analysis of performance bug-

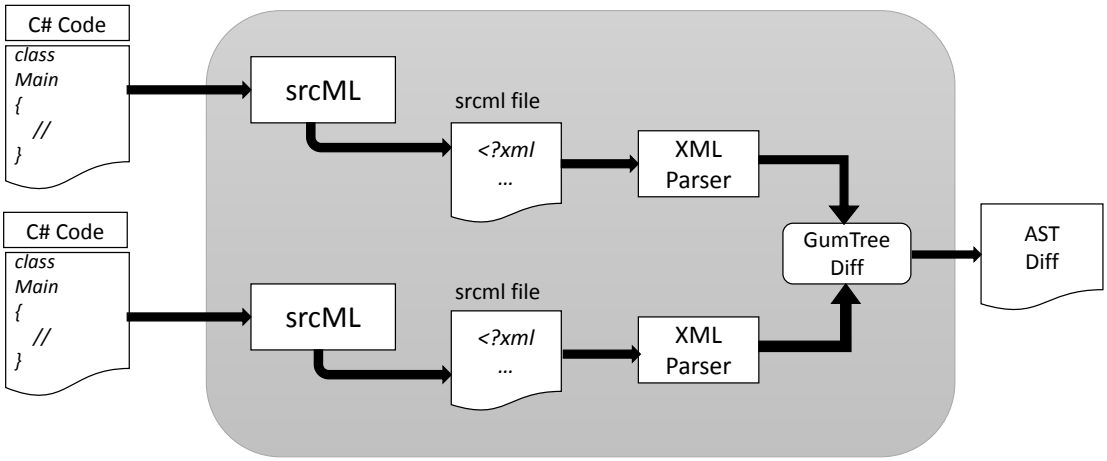

Figure 3.2: C# AST Level Diff Generation

Example 1 C# Code Differencing Output Format

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><patch id="3">
      <classname name="Answer">
            <funcname name="">
                  <stmt id="1">
                        <exp id="0">
                              <type>using name operator name </type>
                              <label> UnityEngine . UI </label>
                              <action>insert</action>
                        </exp>
                  </stmt>
            </function>
      </classname>
</patch>
```

prone methods. Unity Callbacks are unity provided callbacks such as Update. Update method is called in every frame if the MonoBehaviour is enabled. MonoBehaviour is the base class from which every Unity script derives. Similarly, Start is another Unity callback. Custom callbacks are event listeners that developers can add from code. User-Defined Methods are other utility or supporting methods that are used in unity scripts. Apart from that, we also performed analysis for most frequently changed methods. During the most common method change analysis, we performed analyses based on only method body changed based analysis and also method body considering call graph-based analysis.

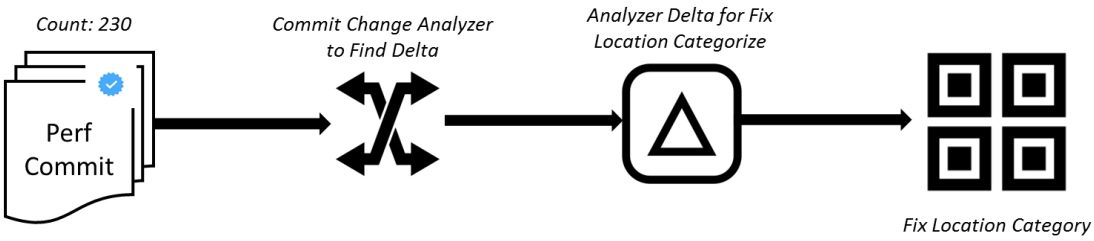

Figure 3.3: Overview of Commit Analysis

3.4 Commit Analysis

In typical bug fix, we do have a normal illusion that performance bugs are mostly fixed in source code. To analyze the factor, we analyzed the performance fix location. So for the 230 performance bugs, we analyzed git commit. We captured the Delta or change in between bug fix commit and its parent commit. From Delta, we identified which files are changed. Based on file type change, we categorized fix location. We did this analysis programmatically with the help of Git Java library JGit. Figure 3.3 shows the overview of commit analysis.

CHAPTER 4: EMPIRICAL EVALUATION

In this section, we describe our settings in Section 4.1, followed by research questions in Section 4.2. Finally, we discuss experiment results in Section 4.3

4.1 Experiment Settings

We collected a dataset of 100 GitHub projects described in Section 3.1. For performance commit identification and code change analysis, we use a computer with 2.4 GHz Intel Core i7 CPU with 16GB of Memory, and Ubuntu 14.10 LTS operating system.

4.2 Research Questions

In our analysis, we seek to answer following research questions.

- **RQ1:** What are the major root causes of *Unity* performance bugs?
- **RQ2:** What are the most common performance bug prone methods of *Unity* performance bugs?
- **RQ3:** How difficult is to fix *Unity* performance bugs?
- **RQ4:** Which files are common to resolve *Unity* performance bugs?

4.3 Results

4.3.1 RQ1:Root Causes of Unity Performance Bugs

With detailed root cause analysis, we identified Unity's performance bug fix taxonomy. That means, we categorized the bugs depending on their types and how they got fixed. We classified these 230 bugs into 15 categories. Figure 4.1 shows the Unity performance bug taxonomy with their frequency. Among these 230 performance bugs, largest category is Asset/Prefab related optimization. In Unity game components are stored in asset file and for reusability some assets

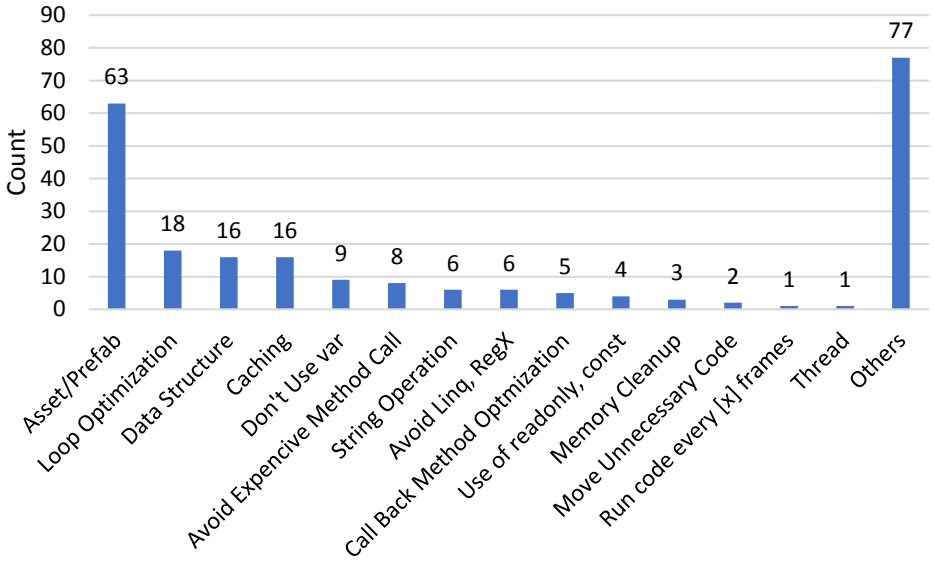

Figure 4.1: Unity Performance Bug Taxonomy with Frequency

are bundled together and used as prefab file. 18 fixes are related to loop optimization and 16 fixes are related to caching related fix. Data Structure related performance fixes accounts for 16 cases. Other fixes are: Don't use var, avoid linq and regx, string operation, use of readonly and constant, Memory cleanup, move unnecessary code, Run code in every X frame and thread related fixes. Performance fixes that we cannot generalized are considered as Others type.

In the following subsections, we will discuss detail about each of the bug category.

Asset/Prefab Bug Fix

The graphical parts of the game can primarily impact two computer systems: the GPU and the CPU. Considering that, a major portion of Unity Performance bugs are related to Asset or Prefab optimization. Example 2 shows such an example of a performance bug fix. In this fix, developers improved performance prefab that is bundling related components as a bundle. We check the unity documentation too that uses of prefab are helpful for performance improvement.

Example 2 Asset/Prefab Related Fix *(AdrianBZG/Medieval_Warfare_VRUnity:207bc51)*

```
+ u1001 & 1246218760
+Prefab:
+ m_ObjectHideFlags:  0
+ serializedVersion:  2
+ m_Modification:
+ m_TransformParent:  fileID: 713704410
```

Loop Optimization

In *Unity* use of typical "for" loop instead of "foreach" helps to improve performance. We also check several Stackoverflow issues to confirm this fix. So, for Unity it's suggested to use "for" rather than foreach.

Example 3 Loop Optimization *(gpvigano/VRTK-GearVR-Test:c2c9e08)*

```
- foreach (VRTK_BasicTeleport teleporter in
    VRTK_ObjectCache.registeredTeleporters)
+ for (int i = 0; i <
    VRTK_ObjectCache.registeredTeleporters.Count; i++)
{
+VRTK_BasicTeleport teleporter =
    VRTK_ObjectCache.registeredTeleporters[i];
teleporter.Teleporting -= new TeleportEventHandler(OnTeleporting)
    ;
```

Caching-Based Performance Fix

Caching is the process of storing data in memory or data structure. In Example 4, rather than loading gameobject again and again. Dictionary is used to save a copy of the already loaded gameobject. This works as a singleton pattern if not created, then create otherwise return existing instance. Caching is widely used in performance optimization of Unity applications.

Example 4 Caching-based Performance Fix *(stefaanvermassen/virtual-museum-app:b3823da)*

```
+private static Dictionary<string, GameObject> objects = new
   Dictionary<string, GameObject>();
...
+ GameObject FastResource(string resource) {
+ if (objects.ContainsKey(resource)) {
+ objects.Add(resource, (GameObject) Resources.Load(resource));
+ }
+ return objects[resource];
+ }
...
GameObject CreateFace(Vector3 localPosition, Vector3 scale,
   Vector3 angles, string type){
...
- var frontSide =
   (GameObject)GameObject.Instantiate(Resources.Load(type));
+ var frontSide =
   (GameObject)GameObject.Instantiate(FastResource(type));
}
```

Data Structure Related Fix

The data structure is a particular way of storing data in a computer. Developers must choose the appropriate data structure for better performance. Example 5 shows a performance fix using right Data Structure. Since this data structure code is always using the first element, so, rather than using a list, the LinkedList is much more effective to get the first item. We got similar issues like using Maps where required.

Don't Use Var

It indicates that it is better to use concrete data type rather than using generic data type var. If the compiler cannot determine exact type of type data pointing to var, then it might create compilation error. Apart from that run time type determination might have performance overhead.

Example 5 Data Structure Related Performance Fix *(stefaanvermassen/virtual-museum-app:b3823da)*

```
- private List<GazeSample> stabilitySamples = new
    List<GazeSample>();
+ private LinkedList<GazeSample> stabilitySamples = new
    LinkedList<GazeSample>();
...

- if (stabilitySamples.Count >= StoredStabilitySamples)
+ // Remove from front items if we exceed stored samples.
+ while (stabilitySamples.Count >= StoredStabilitySamples)
                                        {
- stabilitySamples.RemoveAt(0);
+ stabilitySamples.RemoveFirst();
                                        }
```

Avoid expensive method call

There are some methods that are considered as expensive by the developers, such as GetComponent. In Unity, it is common is to call GetComponent to access components. In many cases, developers call GetComponent method inside Update callback to extract components and pass it other methods as parameter. To improve the performance, we can extract all the components in Start callback and reuse the components. This will reduce the repitative GetComponent method call.

Avoid Linq, Regx

LINQ is a structured query syntax built in C# to retrieve data from different types of data sources. Regx represents regular expressions. It is suggested to avoid Linq and regx.

String Operation

For string concatenation, string boxing, we had to use string builder rather than string append function. In C#, Strings are immutable, which means that after creation if the value is changed it allocate new memory space. Repetitive String value update, concatenation creates lot of garbage. To improve the performance, we should cut down on unnecessary string creation and manipulation.

If String value is determined during runtime, it's better to use StringBuilder class. Similarly, we should avoid boxing such as String.Format() method that takes a String and an object parameter. Boxing creates garbage because of what happens behind the scenes. When a value-typed variable is boxed, Unity creates a temporary System.Object on the heap to wrap the value-typed variable. Boxing might create unnecessary memory overhead.

Callback method optimization

We should remove any expensive calculation and unnecessary things from callback methods. Also, we should remove any empty callback method. Update(), LateUpdate(), and other event functions look like simple functions, but they have a hidden overhead. They do have hidden overhead communication between Unity engine code and managed code. Apart from that, Unity performs additional safety checking before initiating these methods. For a single method, the overhead might not be noticeable, but accumulatively it can create performance overhead.

Use of readonly, constant

4 fix indicates that using readonly, constant improves performance. The use of readonly generates immutable data structures. Immutable data structures cannot be changed once constructed. This makes it very easy to reason about the behavior of a structure at runtime. A const can be optimized by the compiler to be inlined into the code.

Memory cleanup

This is something about cleaning up the game object/ data structure in the destructor. When the variable goes out of scope, garbage collector identifies and deallocates unused heap memory. The garbage runs behind the process to clean up memory periodically. But for the case where Unity developers allocated large block of memory, it might take time to deallocate the memory by the garbage collector. So, to improve the performance, it's better to deallocate memory in the intended time.

Move Unnecessary Code

Writing efficient code and structuring it wisely can lead to improvements in Unity application performance. Loops can be one area where developers can rule out unnecessary code from the loop. In that line choosing wise loop condition and if a code is not required execute in a loop we should move the code outside the loop. Also, by using "if-else" we can stop the execution of some unnecessary code.

Run code every [x] frame

If a code block needs to execute frequently and triggered by an event, it may not be necessary to run every frame. In Example 6, ExampleExpensiveFunction method is called in every frame and it might be costly to execute the program. In case if the method is not mandatory to execute in every frame, then it might be helpful by executing in some interval. Example 7 shows such an optimized version of code where ExampleExpensiveFunction method is called in every 3 frames.

Example 6 Run Code Everytime Example

```
void Update()
{
   ExampleExpensiveFunction();
}     }
```

Example 7 Run Code Every [3] Frame Example

```
private int interval = 3;

void Update()
{
   if (Time.frameCount % interval == 0)
   {
      ExampleExpensiveFunction();
   }
}
```

Thread

In multi-threading program two of multiple parts can run concurrently and each code block can handle a different task at a time. In threading based optimization programs can load game components in multiple threading or render in multiple threads. In some cases optimization can be done through making any thread asynchronous from synchronous to reduce thread waiting time.

4.3.2 RQ2:Bug Prone Methods

Based on our code change diff analysis discussed in Section 3.3, we identified which methods are changed. We categorized these methods as Unity Callback, Custom Callback, and User Defined Method. Unity Callbacks are unity provided callbacks such as Update. Update is called in every frame if the MonoBehaviour is enabled. MonoBehaviour is the base class from which every Unity script derives. Similarly, Start is another Unity callback. Custom callbacks are event listeners that developers can add from code. User-Defined Methods are other utility or supporting methods that are used in unity scripts. Figure 4.2shows the frequency of each type of method in our diff based change analysis. Based on our analysis, Unity callback is changed 224 times, while custom callbacks are changed 409 times. In the case of user-defined methods, those are changed 2706 times.

Apart from that, we identified the top 10 frequently changed methods. If any method body is altered, we considered that the method is changed. As shown in the Figure 4.3, Update method changes most frequently. Others are Start, Awake, OnEnable, etc. From the analysis, we identified that unity callbacks are related to performance bug locations. So, these methods are updated frequently. In this analysis, we did not consider callgraph dependency.

Later we did another analysis considering the code change method and static call graph dependency. According to the call graph figure, if a method is changed, then dependent methods are also considered as changed due to caller callee relationship. When we consider call graph dependency, we identified that the frequency of Unity callbacks also increased. That indicates that user-defined methods that are called from Unity callbacks are prone to performance bugs. Figure 4.4 shows the

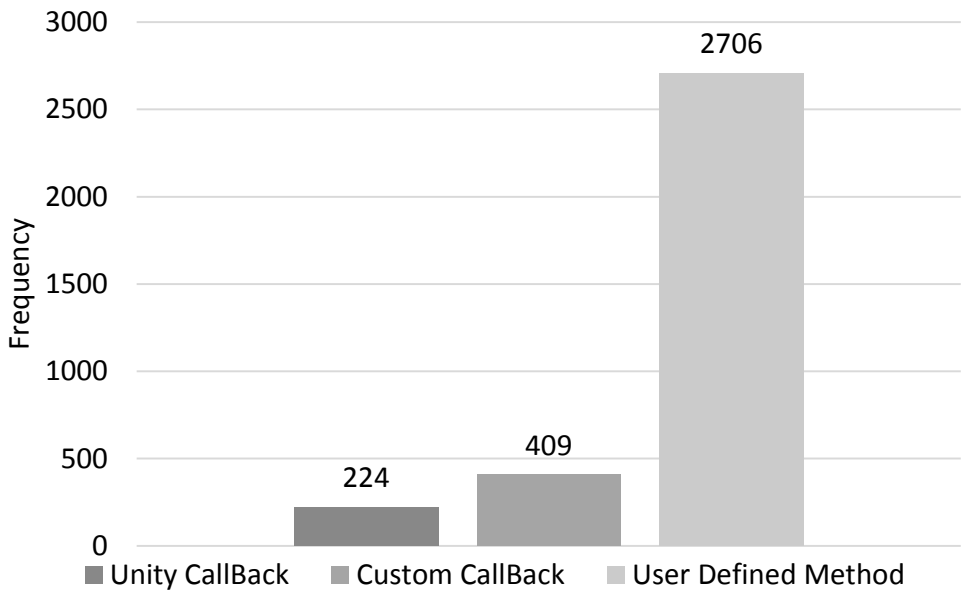

Figure 4.2: Type of Methods Prone to Performance Bug

frequency of Method change frequency based on code change and call graph dependency.

If we consider distinct method changes contributed to 230 performance bugs, for 77 bugs, there is a change in the Update method; for 41 bugs, Start method is changed. OnInspectionGUI method changed in 39 bugs. Like other analyses, we also found the majority of the bugs fixes are related to callback methods. Figure 4.5 shows the top 10 methods that contributed to performance bug fix.

4.3.3 RQ3:Performance Fix Complexity

Based on code change, we measured the Performance Fix Complexity. In terms of class change, the average class change per performance bug is 3.73, and the median is 1.00. For the case of Method-Change, the size average method change is 12.80 method, and the median is 3.50. For statement change average statement change count is 32.83, and the median is 10.50. Figure 4.5 shows the performance fix complexity distribution. From this analysis, we can confer that performance fixes are complex in nature.

Here some of the reasons why performance fixes are complex in nature. The most common reason is code and component dependency. So, if we change in one method or class, then in most cases, it's required to change in code dependency. Another reason we observed that in many cases,

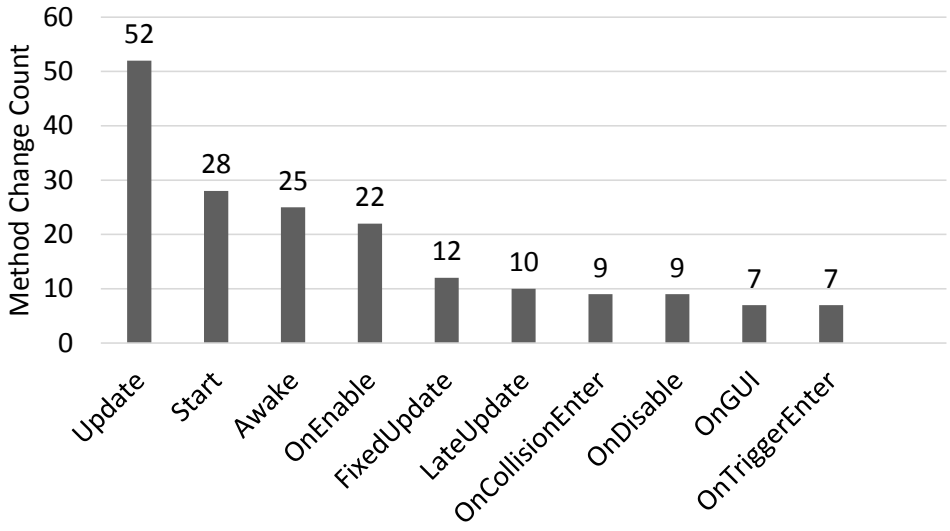

Figure 4.3: Fix Methods Without Considering Call Graph

developers fix several performance issues altogether. For example, we found that when performing loop optimization, developers optimize loop related fixes in several locations. Two other reasons are lack of tool support and documentation. So, performance issues cannot be identified early. Later based on block or StackOverflow, developer refactors code. Since issues are identified later, code refactoring size becomes large.

4.3.4 RQ4:Performance Bug Fix Location

Based on the analysis of change(see Section 3.4), here is the categorized result of performance bug fix location. As shown in Figure 4.7, 92 bugs are fixed by changing only in source code. Forty-five bugs require a change in unity asset or prefab file change. But 93 bugs required to change in both source and unity asset/prefab files. This is critical findings that, to improve or fix performance in many cases, we need to change asset/prefab files. So, the performance analysis of asset/prefab is also important.

Example 8 Example of Asset/Prefab Related Fix *(RussellXie7/Unity_Hololens_Dev:b3823da)*

```
-o.pos = mul(UNITY_MATRIX_MVP, v.vertex);
+o.pos = UnityObjectToClipPos(v.vertex);
```

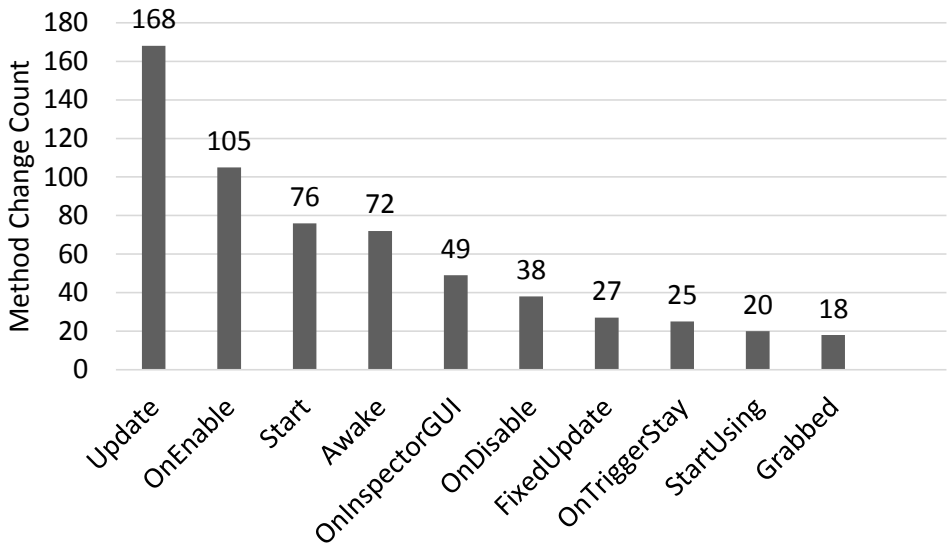

Figure 4.4: Fix Methods Considering Call Graph

Example 8 shows such a performance fix where UnityObjectToClipPos routine is used to calculate Transformation of plane. Unity documentation also mentioned that mul routine could also be used for the Transform plane, but UnityObjectToClipPos is a more efficient way to calculate the plane.

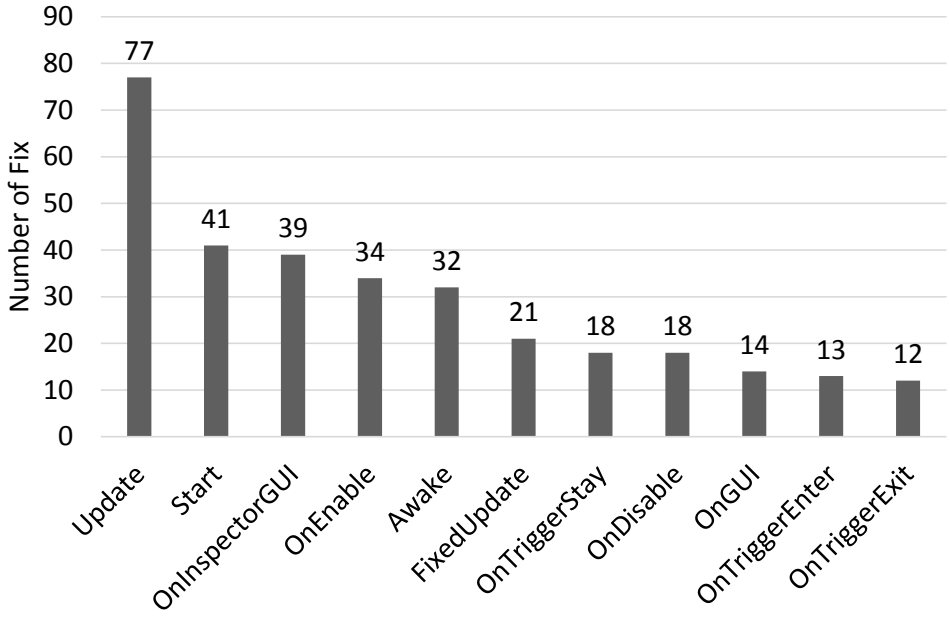

Figure 4.5: Individual Method Contribution to Fix Performance Bugs

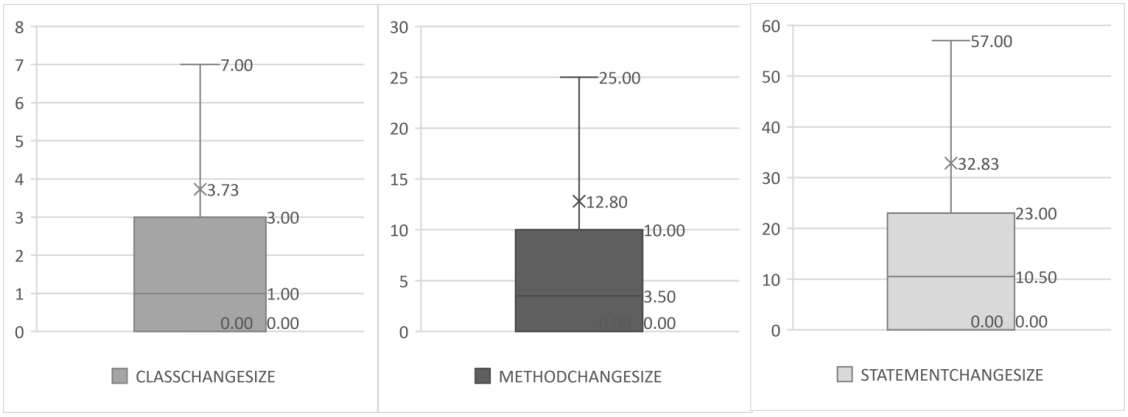

Figure 4.6: Performance Fix Complexity Distribution

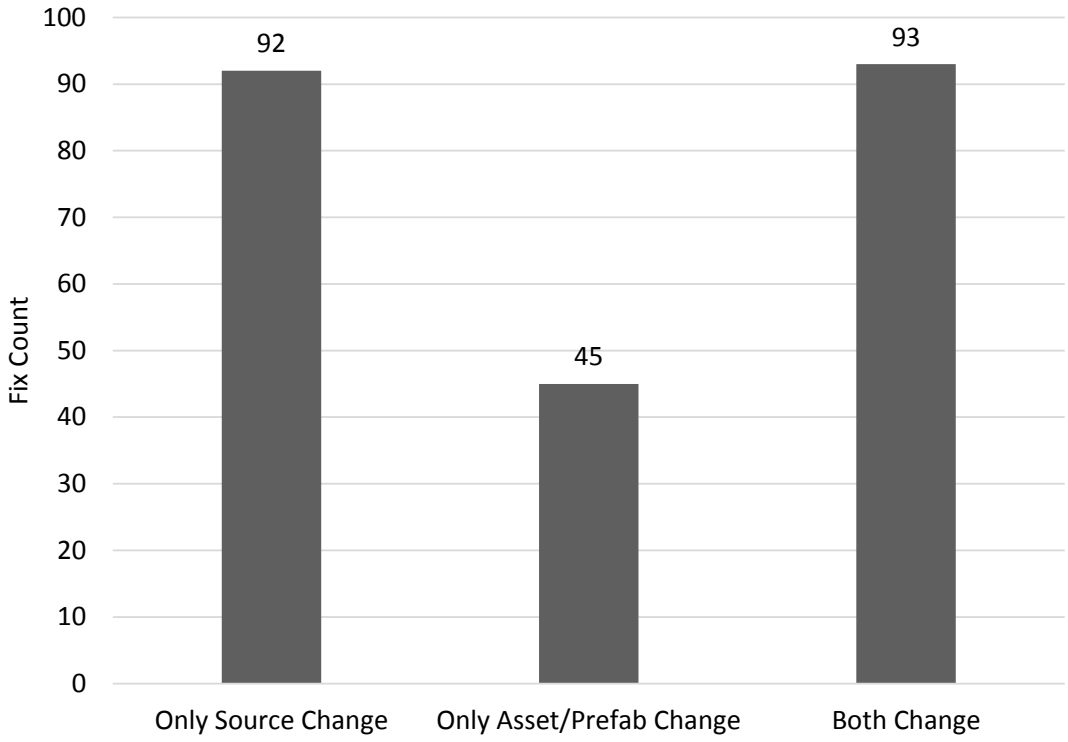

Figure 4.7: Performance Bug Location

CHAPTER 5: DISCUSSIONS

5.1 Threats of Validity

The major threat to the internal validity of our analysis is the correctness of manual categorization of performance bug types. To reduce this threat, we carefully performed all these processes and cross-checked fix types in StackOverflow and Unity documentation for the correctness. The major threat to the external validity of our analysis is that our findings may be project-specific of our analysis dataset. To reduce this threat, we used projects with sufficient commit history and are well maintained.

5.2 Take Away from the Analysis

Unity performance bug taxonomy shows different bug categories that can be useful for bug detection and repair suggestions. Our analysis also identifies several programming language features such as using readonly, const that can be helpful for performance improvement. AST-based diff analysis identifies that Unity callback methods are prone to performance bugs. So, developers should give extra attention while adding code to the callback methods. Code change empirical data also indicates that Unity performance bugs are complex in nature and require to change in multiple code locations. Finally, commit analysis identifies that performance bugs can happen due to Unity GameObject Asset/Prefab Issues.

CHAPTER 6: RELATED WORK

6.1 Empirical Studies of Performance Bugs

Since performance is one of the critical non-functional requirements of software systems, a number of studies analyzed the performance bug of software systems. Zaman et al. [47] did a bug analysis of generic bugs collected from Firefox web browser. Their study focused on security and performance bugs of Firefox web browser. Their followup work [48] analyzed the bug reports of Firefox and Chrome to differentiate characteristics performance and non-performance bugs. To identify performance bug code patterns, Jin et al. [23] did root cause analysis of 109 performance bug collected from five projects. Nistor et al. [38] did a study on performance and non-performance bugs from three popular codebases: Eclipse JDT, Eclipse SWT, and Mozilla. Recent study [15] on 700 performance bug fixing commits across 13 popular open-source projects characterizes the relative frequency of performance bug types as well as their complexity. Prior works on performance bugs focused on generic performance bug categorization. But in our study, we tried to focus on Unity performance bug taxonomy and complexity that can be helpful for developers and further research on identifying performance bugs.

6.2 Empirical Studies of Generic Bugs

There are several research works [18,31,40] that study and characterize different aspects of generic bugs. Park et al. [39] did a study focusing on bugs that need more than one fix attempt. Asaduzzaman et al. [10] mined the bug introducing changes in the Android platform to identify problematic changes. Aranda et al. citeAranda2009 did a study on common bug fixing coordination patterns and to provide implications for software engineering tool developers.

6.3 Performance Bug Detection

Since performance issues are difficult to detect, several research works focused on identifying performance bugs. Several techniques [24, 42, 46] adopted to detect performance bugs. Recent work [36] focused on regression performance test case selection to detect bugs early. Han et al. [20] utilize Machine Learning to generate test frames for guiding actual performance test case generation. While Chen et al. [14] discussed performance Anti-Patterns for applications.

6.4 Performance Optimization of Game Applications

Since Game applications constantly use CPUs and GPUs for frame rendering, the research community focused on optimizing the rendering workload to get a satisfactory performance. DJay [19] utilized a predictive SSIM-based approach to tune GPU workloads. The system used a cloud gaming server that defines cost in terms of GPU time and benefits in rendering quality. Apogee [41] utilizes a low-overhead caching approach that adapts to address patterns found in graphics memory. Kwon et al. [28] proposed a streaming-based execution offloading framework to maximize the uses of computational resources to improve graphics rendering. Vatjus-Anttila et al. [43] built a power model based on rendering complexity such as number of triangles, render batches etc. RAVEN [21] utilizes human visual perception of graphics changes to reduce power uses without degrading user experiences. Prior works mostly focused on graphics rendering and power consumption optimization to improve game applications. In our research, we focused on code optimization to improve rendering based applications such as AR, VR, or Game applications.

CHAPTER 7: CONCLUSION AND FUTUREWORK

In this study, we did an empirical analysis on *Unity* performance bugs derived from 230 performance bug fixing commits across 100 open source projects written in Unity. We manually investigated these commits to confirm that the fixes are related to Unity performance bugs. In summary, we generated Unity performance bug taxonomy and Unity callback methods prone to performance bugs. Apart from that, our analysis performance bugs fix complexity identifies that *Unity* performance bugs are complex in nature and require to change in multiple code locations. Our commit change analysis also figured out that performance bugs can happen due to Unity GameObject Asset/Prefab Issues. As far as our knowledge, this is the first work on Unity performance bugs and this work will be helpful for developers and researchers to identify performance issues.

In the future, we plan to generate a taxonomy of Unity asset or prefab related performance issues. Based on our current findings of code related performance bugs and asset-related finding, we plan to develop an analyzer to detect performance-related bugs. We also planned to build a tool to provide performance bug fix suggestions.

Appendix

Dataset and Source Code: https://github.com/Fariha1502/unityperfanalyzer

BIBLIOGRAPHY

[1] Features and description of unity3d. https://unity3d.com/unity, 2020. Accessed: 2020-06-30.

[2] Jgit - java implementation of the git version control system. https://www.eclipse.org/jgit/, 2020. Accessed: 2020-06-30.

[3] srcml - an infrastructure for the exploration, analysis, and manipulation of source code. https://www.srcml.org/, 2020. Accessed: 2020-06-30.

[4] Unity-asset workflow. https://docs.unity3d.com/560/Documentation/Manual/AssetWorkflow.html, 2020. Accessed: 2020-07-20.

[5] Unity-creating and using scripts. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html, 2020. Accessed: 2020-07-20.

[6] Unity documentation - 2d or 3d projects. https://docs.unity3d.com/, 2020. Accessed: 2020-06-30.

[7] Unity-gameobjects. https://docs.unity3d.com/560/Documentation/Manual/GameObjects.html, 2020. Accessed: 2020-07-20.

[8] Unity-prefabs. https://docs.unity3d.com/Manual/Prefabs.html, 2020. Accessed: 2020-07-20.

[9] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 298–308, USA, 2009. IEEE Computer Society.

[10] M. Asaduzzaman, M. C. Bullock, C. K. Roy, and K. A. Schneider. Bug introducing changes: A case study with android. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 116–119, 2012.

[11] Jae-Hwan Bae and Ae-Hyun Kim. Design and development of unity3d game engine-based smart sng (social network game). *International Journal of Multimedia and Ubiquitous Engineering*, 9(8):261–266, 2014.

[12] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*, volume 2. Prentice Hall Upper Saddle River, 2003.

[13] Ismail Buyuksalih, Serdar Bayburt, G. Buyuksalih, A. Baskaraca, Hairi Karim, and A. Rahman. 3d modelling and visualization based on the unity game engine - advantages and challenges. volume IV-4/W4, pages 161–166, 11 2017.

[14] Tse-Hsun Chen, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. Detecting performance anti-patterns for applications developed using object-relational mapping. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1001–1012, New York, NY, USA, 2014. Association for Computing Machinery.

[15] Y. Chen, S. Winter, and N. Suri. Inferring performance bug patterns from developer commits. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 70–81, 2019.

[16] Daniel V de Macedo and Maria Andréia Formico Rodrigues. Experiences with rapid mobile game development using unity engine. *Computers in Entertainment (CIE)*, 9(3):1–12, 2011.

[17] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 313–324, 2014.

[18] P. Fonseca, Cheng Li, V. Singhal, and R. Rodrigues. A study of the internal and external effects of concurrency bugs. In *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pages 221–230, 2010.

[19] Sergey Grizan, David Chu, Alec Wolman, and Roger Wattenhofer. Djay: Enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit gpu load balancing. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 58–70, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Xue Han, Tingting Yu, and David Lo. Perflearner: Learning from bug reports to understand and generate performance test frames. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 17–28, New York, NY, USA, 2018. Association for Computing Machinery.

[21] Chanyou Hwang, Saumay Pushp, Changyoung Koh, Jungpil Yoon, Yunxin Liu, Seungpyo Choi, and Junehwa Song. Raven: Perception-aware optimization of power consumption for mobile games. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, pages 422–434, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and detecting real-world performance bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 77–88, New York, NY, USA, 2012. Association for Computing Machinery.

[23] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. Understanding and detecting real-world performance bugs. *SIGPLAN Not.*, 47(6):77–88, June 2012.

[24] Charles Killian, Karthik Nagaraj, Salman Pervez, Ryan Braud, James W. Anderson, and Ranjit Jhala. Finding latent performance bugs in systems implementations. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 17–26, New York, NY, USA, 2010. Association for Computing Machinery.

[25] Chung Hwan Kim, Junghwan Rhee, Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Perfguard: binary-centric application performance monitoring in production environments. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 595–606, 2016.

[26] Sung Kim, Hae Jung Suk, Jeong Kang, Jun Jung, Teemu Laine, and Joonas Westlin. Using unity 3d to facilitate mobile augmented reality game development. pages 21–26, 03 2014.

[27] Anil Koyuncu, Kui Liu, Tegawendé F Bissyandé, Dongsun Kim, Jacques Klein, Martin Monperrus, and Yves Le Traon. Fixminer: Mining relevant fix patterns for automated program repair. *Empirical Software Engineering*, pages 1–45, 2020.

[28] Donghyun Kwon, Seungjun Yang, Yunheung Paek, and Kwangman Ko. Optimization techniques to enable execution offloading for 3d video games. *Multimedia Tools Appl.*, 76(9):11347–11360, May 2017.

[29] Owolabi Legunsen, Farah Hariri, August Shi, Yafeng Lu, Lingming Zhang, and Darko Marinov. An extensive study of static regression test selection in modern software evolution. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 583–594, 2016.

[30] OndV rej Lhoták. Comparing call graphs. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 37–42, 2007.

[31] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 329–339, New York, NY, USA, 2008. Association for Computing Machinery.

[32] Qi Luo, Kevin Moran, and Denys Poshyvanyk. A large-scale empirical comparison of static and dynamic test case prioritization techniques. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 559–570, 2016.

[33] Matias Martinez and Martin Monperrus. Coming: A tool for mining change pattern instances from git commits. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 79–82. IEEE, 2019.

[34] Farouk Messaoudi, Gwendal Simon, and Adlen Ksentini. Dissecting games engines: The case of unity3d. In *2015 international workshop on network and systems support for games (NetGames)*, pages 1–6. IEEE, 2015.

[35] I MOLYNEAUX. The art of application performance testing: Help for programmers and quality assurance.[sl]:" o'reilly media, 2009.

[36] Shaikh Mostafa, Xiaoyin Wang, and Tao Xie. Perfranker: Prioritization of performance regression tests for collection-intensive software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, pages 23–34, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Vincenzo Musco, Martin Monperrus, and Philippe Preux. A large-scale study of call graph-based impact prediction using mutation testing. *Software Quality Journal*, 25(3):921–950, September 2017.

[38] Adrian Nistor, Tian Jiang, and Lin Tan. Discovering, reporting, and fixing performance bugs. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 237–246. IEEE Press, 2013.

[39] J. Park, M. Kim, B. Ray, and D. Bae. An empirical study of supplementary bug fixes. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 40–49, 2012.

[40] Swarup Kumar Sahoo, John Criswell, and Vikram Adve. An empirical study of reported bugs in server software with implications for automated bug diagnosis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 485–494, New York, NY, USA, 2010. Association for Computing Machinery.

[41] Ankit Sethia, Ganesh Dasika, Mehrzad Samadi, and Scott Mahlke. Apogee: Adaptive prefetching on gpus for energy efficiency.

[42] N. Siegmund, S. S. Kolesnikov, C. KÃ€stner, S. Apel, D. Batory, M. RosenmÃŒller, and G. Saake. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 167–177, 2012.

[43] J. M. Vatjus-Anttila, T. Koskela, and S. Hickey. Power consumption model of a mobile gpu based on rendering complexity. In *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 210–215, 2013.

[44] Wikipedia contributors. Call graph — Wikipedia, the free encyclopedia, 2020. [Online; accessed 21-July-2020].

[45] Murray Woodside, Greg Franks, and Dorina C Petriu. The future of software performance engineering. In *Future of Software Engineering (FOSE'07)*, pages 171–187. IEEE, 2007.

[46] Dacong Yan, Guoqing Xu, and Atanas Rountev. Uncovering performance problems in java applications with reference propagation profiling. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 134–144. IEEE Press, 2012.

[47] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. Security versus performance bugs: A case study on firefox. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 93–102, New York, NY, USA, 2011. Association for Computing Machinery.

[48] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. A qualitative study on performance bugs. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 199–208. IEEE Press, 2012.

VITA

Fariha was born and brought up in Bangladesh, a small, beautiful country in South Asia. She earned her bachelor degree in Computer Science and Engineering from the Military Institute of Science and Technology(MIST), Bangladesh. She enrolled in the Master's program at The University of Texas at San Antonio (UTSA) in Fall 2017. As part of the Master's thesis, she started working under the supervision of Dr. Xiaoyin Wang. Her main research interest is in the performance optimization of AR and VR applications. She has been awarded UTSA Computer Science award, a competitive scholarship for her outstanding results.