ANALYSIS OF THE ERRORS CAUSED BY THE FRAGMENTATION OF THE ANDROID ECOSYSTEM: AN EMPIRICAL STUDY

A Thesis
Presented to
The Academic Faculty

by

Martin A. Prammer

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science in Computer Science in the
School of Computer Science

Georgia Institute of Technology
May 2019

ANALYSIS OF THE ERRORS CAUSED BY THE FRAGMENTATION OF THE ANDROID ECOSYSTEM: AN EMPIRICAL STUDY

Approved by:

Dr. Alessandro Orso, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Qirun Zhang
School of Computer Science
Georgia Institute of Technology

Date Approved: May 1, 2019

ACKNOWLEDGEMENTS

I wish to thank Professor Alessandro Orso. who oversaw all my undergraduate research and without whom none of which would have been possible.

I also wish to thank my graduate student advisor Mattia Fazzini (PhD candidate), whom I worked under and with throughout my two years of undergraduate research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ADB	Android Debug Bridge
API	Application Programming Interface
APK	Android Package
AWS	Amazon Web Services
CTS	Compatibility Test Suite

CHAPTER 1

ABSTRACT

Software testing and debugging has always been a pervasive problem for software developers. Mobile applications are highly important to our lives and ensuring their correctness is challenging problem. Android is a popular platform for both developers and users as there are many kinds of devices that can run the operating system. However, because of the highly fragmented nature of the Android ecosystem, it is a complex task to verify that apps behave as expected. To provide more insight into this problem, we performed a study to learn quantitative information about the problems caused by fragmentation. We conducted our study by leveraging cloud-based testing services with existing and suitably developed test suites. We implemented this study by utilizing the Amazon Web Services Device Farm and Android Compatibility Test Suite to execute these tests on a large scale. As a preliminary study, we have focused on a subset of the Compatibility Test Suite test packages and have classified the discovered test failures. We present the results of our study and the fragmentation issues discovered, which we release to assist developers and device vendors in accounting for fragmentation inconsistencies. In future work, we see this study acting as a foundation for continued quantitative analysis on the fragmentation within the Android ecosystem.

CHAPTER 2

INTRODUCTION

We use mobile applications (or simply apps) for a variety of daily activities, including tasks such as online shopping or news consumption. However, while individuals may be using the same apps, they may be using different hardware or operating system versions. These individual components may cause inconsistent behavior through compatibility issues. These compatibility issues exist on a massive scale within the Android platform. However, because of this scale, manually testing for these issues is prohibitive [1].

The Android Compatibility Test Suite (CTS) tests a wide variety of common platform[1] features by testing the Android API [2]. This is accomplished by evaluating the behavior of the Android API implemented on a phone and verifying that it is compatible with the API specification. Common software tests may verify that an Android API method behaves correctly, such as properly throwing an error when given an invalid input. Likewise, hardware tests may explore possible differences between claimed and demonstrated phone features, like testing camera resolution. As a whole, the test suite provides a quantitative analysis of how correct a device's Android API implementation is. Ensuring that the Android API behaves correctly on a device helps developers ensure that their apps behave correctly.

By running tests from the Android CTS, we can characterize inconsistencies on an Android platform. To address the prohibitive nature of manually testing for these issues, we leverage a cloud-based testing service to access each individual Android platform. We utilized the Amazon Web Services (AWS) Device Farm, which has become

---

[1] A platform is the combination of hardware and operating system used to run an app.

a useful tool for developers interested in testing their applications on physical devices [3]. The usage of the AWS Device Farm allows for the execution of tests on real devices without purchasing a physical device by renting temporary device access.

We utilize the AWS Device Farm and the Android CTS to conduct our study, enabling us to collect CTS results from a large set of Android platforms.

This thesis makes the following contributions:

1. We contribute our study and produced results, from running an Android CTS test package on the AWS Device Farm.
2. We make our infrastructure to replicate the study publicly available.

CHAPTER 3

MOTIVATING EXAMPLE

In this chapter, we identify the motivating example for our study.

A bug reported to a developer might not be caused by a mobile application, but instead an issue within the Android platform a specific user operates. For example, there may be a bug in the operating system of a user device [4]. When Samsung provided an update for their Android 4.2.2 phones, the “appcompat” library was not properly upgraded. This caused the affected devices to produce errors and crash apps when these apps utilized the affected part of the support library. Included in the affected components was the support Action Bar, a widely used GUI element. This caused large amounts of confusion for developers, as their apps were now crashing due to errors in their dependencies. This error was fixed when the device vendors applied the next appcompat library update. However, affected devices still exist, as not all users will update their phones. Due to the nature of this bug, it is near impossible for developers to test their devices for this crash without either already knowing about this crash or being able to leverage large scale testing platforms with multiple firmware revisions of selected devices.

With this motivation, our technique uses the Android CTS to test for possible fragmentation errors between each tested device and the Android API.

CHAPTER 4

STUDY

In this chapter, we describe the design and implementation of our study.

To examine issues of fragmentation, we conducted a study using the AWS Device Farm to run Android CTS tests. The outputs from every run are aggregated to produce our results. We examine these results to identify fragment inconsistencies for each device. We used the CTS "ViewTests" package, because of the previously discussed appcompat issue in mind. We find using this test package compelling as the functionality to display content to the screen is a widespread feature of mobile applications.

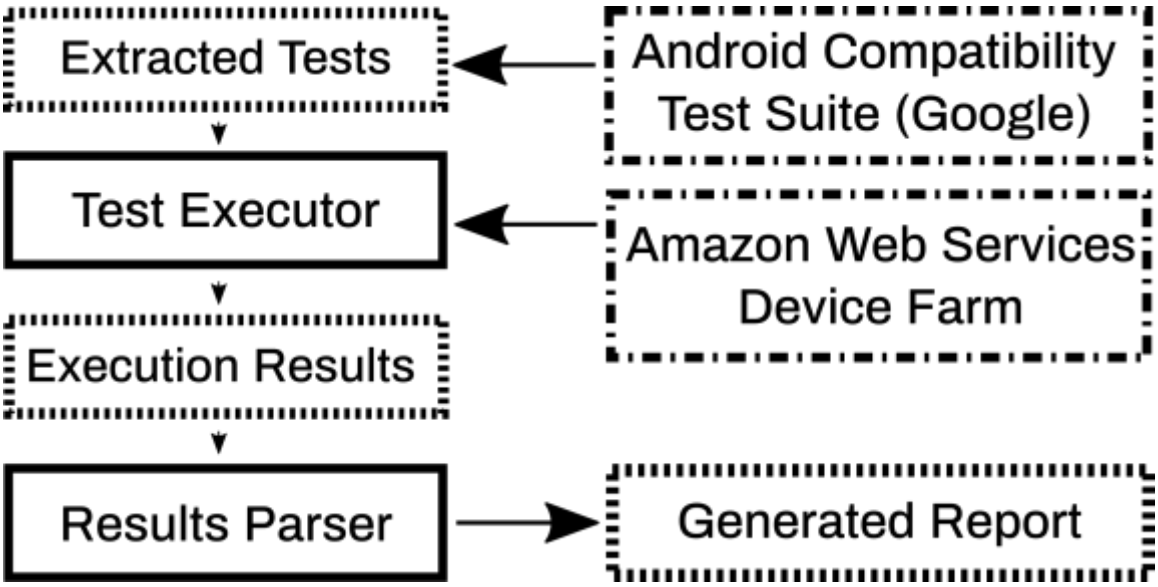

Figure 1. Workflow of the study

To facilitate our study, we implemented two individual modules: The Test Executor and The Results Parser. The Test Executor prepares the required materials to run an Android CTS test on the AWS Device Farm, schedules the test to be run, and then

downloads the results as they are completed. The Results Parser uses these results to build the generated test report.

4.1 Test Executor

The Test Executor prepares the test environment for the Device Farm. This includes a “stub application” and “test application”, which together act as a mobile app that can run tests. The stub application is like a standard mobile application. The test application contains a set of Instrumented JUnit 4 test cases, which are run on the stub application [5], [6]. The stub and test application APK(s) are uploaded using the AWS Command Line Interface tool as part of the environment setup process. In addition, we provide a “custom environment” with each of the scheduled tests, that facilitates the following purposes:

1. Our custom environment collects the device and AWS Device Farm profile information, including a “fingerprint” that contains hardware and operating system version and revision information. We collect this information to identify the state of the specific fragments on the device.
2. Our custom environment ensures the proper installation of the stub and test applications. We do this to verify that all permissions requested by the tests are explicitly granted.
3. The environment configures the device to match CTS requirements, such as setting the system clock to 12-hour representation.
4. The environment specifies additional logging outputs, including saving a copy of the Android Instrumentation file for later parsing.

The environment specifies the behavior of the machine hosting the tests, where the test host executes the required commands using the ADB. The uploaded resources are installed onto candidate devices by scheduling each test run, which instructs the test host to begin the tests. Once the tests are completed, the results are downloaded when signaled

and made available by the Device Farm. All raw artifacts produced by both the tests and environments are downloaded once the tests are reported as completed by the Device Farm.

Each scheduled test run is repeated three times to ensure the validity of the results.

4.2 Results Parser

The Results Parser takes in the artifacts generated by the Test Executor and builds a general test report from the test executor's run. One of the files produced as part of the results is the Instrumentation file, which details the results of every test. Using these results, we classify each test with the following labels:

- "Passed": A test successfully ran to completion.
- "Failed": A test did not successfully run to completion due to a deliberately thrown exception in the test, such as a variable having an incorrect value.
- "Errored": A test did not successfully run to completion due to an unexpected exception in the test, such as not being able to connect to a service that is assumed to be working. This immediately halts the test instrumentation, causing all successive tests to be "Skipped."
- "Skipped": A test was not run as a previously run test "Errored."

The collected results showcase compatibilities between devices, test versions, and API features, where tests that pass a specific CTS test are compliant with the Android API specification for the tested feature. To accompany each test result, the profile information is aggregated into both a device and test host profile, making execution specific information such as device firmware revisions available as part of the test report. All of this generated information is then aggregated into the singular test failures report, providing a high-level view of the device and platform compatibility for the tested feature sets between versions and any errors or inconsistencies present.

Each of the unified test results per device, test, and run are then aggregated to form a final report. We use the following logic to classify our results:

- If a test has at least one “passed” result, it is classified as “passed.” This is because some tests are not consistent in their results, where a passing test may fail at times.
- A “skipped” result is interpreted as a “cannot determine” result, as we cannot determine if the test would have failed for this run.
- If a test is either “errored” or “failed” on all test runs, it is classified as either “errored” or “failed” respectively. A test cannot be classified as “errored” or “failed” if it has at least one “passed” or “cannot determine” result.
- If a test has both “errored” and “failed” results, it is counted as “errored” to preserve the validity of reported failures.

CHAPTER 5

STUDY RESULTS

In this chapter, we report the results for the Android CTS executions on the AWS Device Farm from our study.

The study was evaluated using the CTS "ViewTests" test package. We present the number of individual tests in each version of this test package as Table I.

Table I. Number of tests per CTS ViewTests version

CTS ViewTests Version	Test Count
4.1	626
4.2	633
4.3	631
4.4	635
5.0	636
5.1	642
6.0	681
7.0	728
7.1	744
8.0	901
9	969

We ran the ViewTests test package on 123 Android devices. In total, we detected 224 failing tests across 72 unique devices. Of these failing tests, there were 26 unique tests that failed. We present this data as Table I and Table II. For both tables, results with a count of 0 are omitted.

Table II. Number of test failures per device

Device (product: model)	Device Version	Test Failure Count
H8416: H8416	9	8
dreamqltesq: SM-G950U	7	5
elsa_att_us: LG-H910	7	5
elsa_tmo_us: LG-H918	7	5
athene: Moto G (4)	7	5
noblelteuc: SAMSUNG-SM-N920A	7	5
gtaxlwifixx: SM-T580	7	5
marlin: Pixel XL	7.1.2	5
ocnwhl_00617: HTC U11	7.1.1	5
TB-8504F: Lenovo TB-8504F	7.1.1	5
greatqlteue: SM-N950U1	7.1.1	5
zeroltexx: SM-G925F	7	4
sailfish: Pixel	7.1.2	4
dreamqlteue: SM-G950U1	8.0.0	4
H8266: H8266	8.0.0	4
walleye: Pixel 2	8.0.0	4
taimen: Pixel 2 XL	8.0.0	4
marlin: Pixel XL	8.0.0	4
starqlteue: SM-G960U1	8.0.0	4
gts3lwifixx: SM-T820	8.0.0	4
G8342: G8342	8.0.0	4
judyln_lao_com: LM-G710	8.0.0	4
j7toplteue: SM-J737U	8.0.0	4
star2qlteue: SM-G965U1	8.0.0	4
blueline: Pixel 3	9	4
crosshatch: Pixel 3 XL	9	4
beyond1qlteue: SM-G973U1	9	4
beyond0qlteue: SM-G970U1	9	4
cingular_us: HTC One_M8	4.4.2	4
Y2_Pro: Aqua Y2 Pro	4.4.2	4
dreamqlteue: SM-G950U1	7	3
elsa_vzw: VS995	7	3
dream2qltesq: SM-G955U	7	3
serranoltevzw: SCH-I435	4.4.2	3
klteattactive: SAMSUNG-SM-G870A	4.4.2	3
hlteuc: SAMSUNG-SM-N900A	4.4.2	3
klteuc: SAMSUNG-SM-G900A	4.4.2	3

hammerhead: Nexus 5	4.4.2	3
d2vzw: SCH-I535	4.4.2	3
thor: KFTHWI	4.4.3	3
g3_att_us: LG-D850	4.4.2	3
t0ltevzw: SCH-I605	4.4.2	3
zerofltetmo: SM-G920T	7	2
e7lte_att_us: LG-V410	4.4.2	2
g3_vzw: VS985 4G	4.4.2	2
jfltevzw: SCH-I545	4.4.2	2
degaswifiue: SM-T230NU	4.4.2	2
wiko: RAINBOW 4G	4.4.2	2
razorg: Nexus 7	4.4.4	2
jflteuc: SAMSUNG-SGH-I337	4.4.2	2
g3_tmo_us: LG-D851	4.4.2	2
serranoltexx: GT-I9195	4.4.2	2
kltetmo: SM-G900T	4.4.2	2
occam: Nexus 4	4.4.4	2
w5_mpcs_us: LGMS323	4.4.2	2
jfltetmo: SGH-M919	4.4.4	2
obake_verizon: XT1080	4.4.4	2
peregrine_att: XT1045	4.4.4	2
e7ltezs: SM-E7000	4.4.4	2
fortuna3gxx: SM-G530H	4.4.4	2
vivalto5mve3gdd: SM-G316HU	4.4.4	2
kltevzw: SM-G900V	4.4.4	2
j1pop3gjv: SM-J110H	4.4.4	2
trlteuc: SAMSUNG-SM-N910A	4.4.4	2
trltevzw: SM-N910V	4.4.4	2
D6603: D6603	4.4.4	2
jflteuc: SAMSUNG-SGH-I337	5.0.1	1
jfltevzw: SCH-I545	5.0.1	1
hammerhead: Nexus 5	6	1
klteuc: SAMSUNG-SM-G900A	6.0.1	1
heroltexx: SM-G930F	6.0.1	1
d2vzw: SCH-I535	4.3	1

Devices are identified by their “product”, “model”, and “version” strings, as retrieved by the ADB as part of the “Device Properties” file. Device versions are reported

separately from device product and model information for clarity. "Test Failure Count" indicates the number of consistent, unique test failures across all test runs.

Table III. Number of failures per failing test

Failing Test	Count
android.view.cts.ViewTest.testGetLocalVisibleRect	34
android.view.cts.ViewTest.testMeasure	34
android.view.cts.SurfaceViewSyncTests.testEmptySurfaceView	16
android.view.cts.SurfaceViewSyncTests.testSmallRect	16
android.view.cts.SurfaceViewSyncTest.testEmptySurfaceView	16
android.view.cts.SurfaceViewSyncTest.testSmallRect	16
android.view.cts.SurfaceViewSyncTest.testSurfaceViewBigScale	16
android.view.cts.SurfaceViewSyncTest.testSurfaceViewSmallScale	16
android.view.cts.SurfaceViewSyncTests.testSurfaceViewBigScale	12
android.view.cts.SurfaceViewSyncTests.testSurfaceViewSmallScale	10
android.view.cts.ViewTest.testFilterTouchesWhenObscured	10
android.view.cts.SurfaceViewSyncTests.testVideoSurfaceViewRotated	5
android.view.cts.DisplayRefreshRateTest.testRefreshRate	4
android.view.cts.SearchEventTest.testTest	4
android.view.cts.SurfaceViewSyncTests.testVideoSurfaceViewCornerCoverage	3
android.view.cts.SurfaceViewSyncTests.testVideoSurfaceViewTranslate	2
android.view.cts.MotionEventTest.testReadFromParcelWithInvalidSampleSize	1
android.view.cts.View_UsingViewsTest.testSetupListeners	1
android.view.cts.TextureViewTest.testFirstFrames	1
android.view.inputmethod.cts.InputMethodManagerTest.testInputMethodManager	1
android.view.cts.SurfaceViewSyncTest.testVideoSurfaceViewCornerCoverage	1
android.view.cts.SurfaceViewSyncTest.testVideoSurfaceViewTranslate	1
android.view.cts.SurfaceViewSyncTest.testVideoSurfaceViewRotated	1
android.view.cts.SurfaceViewSyncTest.testVideoSurfaceViewEdgeCoverage	1
android.view.inputmethod.cts.InputMethodInfoTest.testInputMethodSubtypesOfSystemImes	1
android.view.inputmethod.cts.InputConnectionWrapperTest.testInputConnectionWrapper	1

"Count" indicates the occurrences of a specific test failing across all devices.

CHAPTER 6

DISCUSSION

In this chapter, we discuss the results of our study. We then evaluate possible threats to the validity of our study and the results.

6.1 Results Analysis

In this section, we discuss the results of our study, and emphasize the results we found particularly important.

The test results identified multiple possible fragmentation inconsistencies, which warrant future manual inspection. We detail our generated report as we see this information beneficial to both developers and platform vendors in accounting for and acting on these inconsistencies. We emphasize two kinds of results:

1. If a test is only failed by a small number of devices, it presents a unique challenge for developers to identify and debug the associated Android API component if a fragmentation issue arises, as only a small number of users will present with this issue.
2. If a device only fails a small number of tests, it presents an equally unique challenge for developers, as an otherwise highly compliant device may only have one or two unique fragment inconsistencies.

Due to their nature, these results warrant more extensive manual investigation into both the causes and impacts of these fragment inconsistencies.

6.2 Threats to Validity

In this section, we discuss the possible threats to the validity of our study, and the decision making regarding each.

Test errors are not counted as test failures. This behavior was deliberately chosen as some test errors were due to the test environment and not the device itself. Specifically, Android 9 devices had regular crashes due to being unable to connect to the Camera Service. While some tests might have been real failures, if they caused the test instrumentation to exit early and skip tests, the test is always classified as an error to assist in preventing the overreporting of test failures.

In each test run, many tests may be skipped due to the test instrumentation exiting early. This occurs because of the test instrumentation crashing during the test and causing the test package to exit early. Because individual test classes may not run in the same order between test runs, the number of skipped tests in a single run will vary between test runs. To prevent overclaiming test failures, we exclude tests that have at least one skipped result. This behavior assists in preventing the overreporting of test failures.

CHAPTER 7

RELATED WORK

In this chapter, we identify related work to our study and identify where our contribution is relevant.

The fragmentation of the Android ecosystem has been a focus of study in literature for several years [7], [4]. Han and colleagues [7] are among the first to study the issues caused by fragmentation. They systematically analyze bug reports related to two popular device vendors by utilizing the top bug reports over time and propose a method for tracking fragmentation. Wei and colleagues [4] propose a technique to identify compatibility issues. One of these issues was used as our motivating example.

Previous studies on the fragmentation and its impact on developers have been conducted as well [8], [9]. Joorabchi and colleagues [8] studied the challenges faced by mobile developers and highlight areas of possible improvement. The lack of information for developers regarding individual platforms is a challenging problem. Our work helps developers by providing a concrete dataset of inconsistencies across many devices. Wu and colleagues [9] studied the impact of vendor customization on Android device security. We believe this line of research is complementary; while we used a test package to validate Android API correctness, utilizing a test package that explores device security would be a broader application of a continuation of our study.

CHAPTER 8

CONCLUSION

In this thesis, we conducted a study to explore fragmentation inconsistencies in the Android ecosystem. We implemented a system to run tests from the Android Compatibility Suite on devices made available on the Amazon Web Services Device Farm. In running the tests, we discovered 26 unique test failures across 72 unique devices. The work in this thesis is an initial step in quantifying the fragmentation of the Android ecosystem. The thesis can be expanded by future work in ways such as:

1. The results could be used to generate a taxonomy, detailing the widespread fragment inconsistencies across the Android ecosystem. This would be useful as a tool to assist developers and users in platform specific debugging. This would also be an effective means of communicating widespread fragmentation issues to device vendors.
2. The results could also be used to assist both developers and researchers in creating techniques to account for fragmentation issues.

REFERENCES

1 OpenSignal. *Android Fragmentation Visualized*. 2015. Website - https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf

2 Android Open Source Project. *Compatability Test Suite*. Website - https://source.android.com/compatibility/cts

3 *Amazon Web Services Device Farm*. Website - https://aws.amazon.com/device-farm/

4 L. Wei, Y. Liu, and S.-C. Cheung, "Taming Android fragmentation:Characterizing and detecting compatibility issues for Android apps," in *2016 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016.

5 Android Open Source Project. *Test your app - Instrumented tests*. Website - https://developer.android.com/studio/test

6 JUnit. *JUnit 4*. Website - https://junit.org/junit4/

7 D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding Android Fragmentation with Topic Analysis of Vendor-Specific Bugs," in *Proceedings of the 2012 Working Conference on Reverse Engineering*, 2012.

8 M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real Challenges in Mobile app Development," in *2013 ACM/IEEE International Symposiumon Empirical Software Engineering and Measurement (ESEM)*, 2013.

9 L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang, "The impact of vendor customizations on android security," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013.