2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing

An Empirical Investigation of Fault Triggers in Android Operating System

Fangyun Qin*, Zheng Zheng*, Xiaodan Li†, Yu Qiao*, Kishor S. Trivedi†
*School of Automation Science and Electrical Engineering, Beihang University, Beijing, China
†Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA
Email: {fangyunqin, zhengz, qiaoyu}@buaa.edu.cn, {xiaodan.li, ktrivedi}@duke.edu

***Abstract*—The growing popularity and complexity of Android operating system makes it prone to suffer failures during usage, which increases difficulties of fixing bugs. Different strategies and mitigation methods can be developed and applied based on different types of bugs, which gives rise to the necessity to have a deep understanding of the nature of bugs in this system. In this paper, an empirical study is taken on 513 bug reports from Android operating system. A bug classification is conducted according to fault triggering conditions, followed by the analysis of bug types and bug attributes. Moreover, the comparison of bug types between Android and Linux is carried out. This paper reveals ten interesting findings based on the empirical results from these three aspects and further provides guidance for developers and users based on these findings.**

***Index Terms*—bug classification; fault trigger; Android; Mandelbug**

I. Introduction

As the number of useful and convenient applications in smartphones is increasing, more and more people prefer to use their smartphones for entertainment and daily activities. Android is the most popular mobile operating system in the world, with hundreds of millions of mobile devices distributed among more than 190 countries [1]. Along with its wide application, many studies concentrate on its security [2], memory [3], power consuming [4], bugs [5], etc. In this paper, we analyze the Android operating system from the viewpoint of bugs.

Previous studies on bugs in Android mainly focused on bug locations in modules and their persistence, code modification [5], duplicate bug detection [6], [7], or bug changes [8], etc. Although the fruits present good results for the understanding of bugs in Android, the distribution of bug types in this operating system and the relationship between the bug types and bug attributes are significant topics deserved to be studied, which has not been fully explored till now.

Several bug classification methods have been conducted based on bug manifestations. For example, Gray [9] classified bugs into Bohrbugs and Heisenbugs according to the principle whether corresponding failures can be systematically reproduced or not. Grottke et al. [10] defined Mandelbugs as the complementary antonym of Bohrbugs. Due to the simplicity of the activation and error propagation of a Bohrbug, its failure reproduction and hence isolating are easy to be processed. Comparatively the activation and/or error propagation of a Mandelbug are complex – the complexity may be induced by the time lag between bug activation and the failure occurrence, interactions of the software application with its system-internal environment, the timing of inputs and operations, and the sequencing of operations. Different from Bohrbugs, Mandelbugs could make system exhibit chaotic or even nondeterministic behavior during operation [11]. Furthermore, Mandelbugs can be divided into aging related Bugs (ARB) and non-aging related Mandelbugs (NAM). ARBs refer to bugs that can result in software aging (a phenomenon that software exhibits an increasing failure rate and/or degrading performance in a long-running software system). Based on above classification, Cotroneo et al. [12] proposed an extensive analysis of fault triggers. Several researchers have analyzed the manifestation characteristics of each type of bugs in popular software systems, like Apache HTTPD server, MySQL, Linux kernel, Apache AXIS framework [12] and space mission system software [13]. To the best of our knowledge, this is the first paper focusing on the bug classification according to fault triggers in Android.

Our study is performed with 513 bug reports of Cyanogenmod, a widely used Android operating system [14]. For each report, we examine the bug description, comments, as well as attached files carefully. The empirical study is undertaken from the following three aspects:

(1) Analyze the bug types in Android. In this part, a systematic capture of bug distribution is obtained. The observations are useful to guide developers in Android focusing on the specific subtypes during bug mitigation.

(2) Explore the relationship between bug types and bug attributes, such as bug priority, fix duration and components. Exploring bug attributes related with bug types enables developers to have a deep understanding of bugs in Android, such as which component should be firstly focused on, whether fix duration of bugs is related with bug types.

(3) Compare bug types between Linux and Android. The comparison can provide a deeper understanding about these two operating systems from the viewpoint of bug types.

Based on the empirical study on above three aspects, the work provides a set of findings and implications summarized in Table I, which can provide helpful guidance for developers and users.

The rest of the paper is organized as follows. Section II presents the related work, and Section III concentrates on the bug source and classification procedure. In Section IV, our

978-1-5090-5652-1/17 $31.00 © 2017 IEEE
DOI 10.1109/PRDC.2017.27

TABLE I
SUMMARY OF FINDINGS AND IMPLICATIONS

Findings on bug types	Implications
(1) About two thirds (65.2%) of the examined bugs are Bohrbugs and the fraction of Mandelbugs is 31.4%.	As to Bohrbugs, the previous testing is insufficient, more unit testing, function testing, as well as integration testing are necessary. As to Mandelbugs, a non-negligible fraction exists. Due to their difficulty in fixing, cost-effective fault tolerance techniques, such as environment diversity should be developed to handle them at run time.
(2) Almost all (97.6%) of the examined NAMs belong to three subtypes: ENV, LAG and NAU. Half of NAMs are of subtype ENV.	To reduce the impact of Mandelbugs in Android, bug detection, testing or fault tolerance should focus on these three major bug subtypes, especially ENV. For ENV, testing under multi-configurations and frequent function switch in an application, and two or more applications run together should be paid attention to.
(3) Most (76.2%) of the examined ARBs are of MEM subtype.	The memory characteristic in Android has been attracting large attention. Developers are suggested to pay special attention to the resource release. Moreover, more performance testing is required to reduce memory leaks.
Findings on bug attributes	**Implications**
(4.1) Bohrbugs are almost uniformly distributed in components, while Mandelbugs are prone to lie in the components of Camera, Audio and Bluetooth. (4.2) Camera and Bluetooth seem to have the same number of Bohrbugs and Mandelbugs.	For detecting Mandelbugs, components Camera, Audio and Bluetooth are suggested to be carefully examined. Meanwhile, components Camera and Bluetooth should be noticed for their large percentage of Mandelbugs.
(5) In Android, it tends to take more time to fix Mandelbugs than Bohrbugs.	For Mandelbugs, due to their long fix duration, other mitigation techniques (such as retry, restart, reboot) can be utilized to prevent their manifestation.
(6) In Android, the bug priority is not influenced by bug type.	For mitigating Bohrbugs and Mandelbugs during design, the relative importance of these two kinds of bugs is prone to be related with their proportions.
Findings on bug types Android vs. Linux	**Implications**
(7) Although Android is developed based on Linux, their bug percentages are quite different.	Although Android is developed based on the Linux kernel, developers could not use the benefits in Linux directly in Android. For mitigating ARBs in Android, Developers can design multi-level software rejuvenation strategies (such as close the application from recent app and restart it, reboot the phone) on the basis of the memory usage to get better user experience.
(8) The percentage of ARBs in Android is almost half of that in Linux.	
(9) Among the subtypes of NAM, the percentage of TIM bugs in Android is significantly less than those in Linux.	Developers do not need to pay too much attention to TIM subtype in Android as that in Linux.
(10) MEM subtype is dominant among ARBs in both Android and Linux.	Refer to implications corresponding to Finding (8).

empirical results are described in detail, followed by threats to validity in Section V. Section VI concludes the paper.

II. RELATED WORK

In the process of software development, bug classification helps in the early identification of types in defect inflow profiles [15]. There are mainly three schemes that are used to conduct bug classification: HP scheme [16], IEEE Std. 1044 [17] and ODC scheme [18]. ODC classified defects according to several attributes, among which defect type and defect trigger are the most important ones. Defect type represents the semantics of the fix and provides feedback on the development process. A programmer usually makes the correction based on it. Meanwhile, defect trigger refers to the conditions that make a defect surface as a failure, which provides feedback on the verification process.

The classification method used in this paper is based on potential manifestation characteristics of a bug and its failure reproducibility. The reproducibility of bugs is first systematically studied in [9], where bugs are classified into Bohrbugs and Heisenbugs. The occurrence of failures due to Bohrbugs is fixed which is considered as solid as Bohr atom, and they are easy to be detected, while Heisenbugs are soft bugs which behave uncertainly when testers attempt to reproduce failures caused by them. Grottke et al. [11] used Mandelbugs as a complementary to Bohrbugs. Their classification was based on the complexity of bug activation and error propagation. Mandelbugs are a more general classification and Heisenbugs are included in Mandelbugs.

Fig. 1. Failure mechanism for ARBs

The definitions of Bohrbug and Mandelbug are as follows:

(1) Bohrbug: a kind of bug whose activation and error propagation are simple, reproducing failures and hence isolating these bugs is easy. If the triggering conditions of the bug are met, the failure will occur definitely.

(2) Mandelbug: a kind of bug which is difficult to be activated and/or has complicated error propagation. The complexity results from either direct or indirect factors. The direct factors refer to a time delay between bug activation to the occurrence of failure. The indirect factors include interactions of software application with its system-internal environment (hardware, operating system, or other applications), the timing of inputs and operations, or the sequencing of different operations.

There is a special subtype of Mandelbug called aging related bug (ARB) as shown in Fig. 1. It is a kind of bug that can

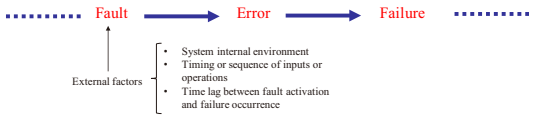

Fig. 2. Failure mechanism for NAMs

lead to software aging. Software aging is a phenomenon in the long-running software system, where software tends to show increasing failure rate and/or degrading performance. ARBs are related to accumulation process. It takes a long time for ARBs to lead to failure due to error accumulations. Apart from some conditions that activate ARB, environment conditions should be met to make the aging related errors propagate [19]. The causes of ARB are mainly memory bloating and leaking, unreleased file-locks, unterminated threads, storage fragmentation, data corruption, accumulation of round-off errors and so on [12]. The rest of Mandelbugs are defined as non-aging related Mandelbugs (NAMs) which are shown in Fig. 2. NAMs are complex due to possible influence of external factors.

Moreover, an extensive bug classification method was proposed in [12] based on different kinds of complexity in fault trigger conditions. This extended previous classification method and gave more accurate information about bug type.

Based on above classification methods, some empirical investigations are conducted. In [13], 520 software bugs in 18 JPL/NASA space missions are analyzed. The research revealed that there were about 36.5% of bugs belonging to Mandelbugs, and 61.4% were Bohrbugs. Moreover, ARBs constituted 4.4% on average. In [12], authors performed an extended bug classification in four open-source software systems: Linux kernel, MySQL, Apache HTTPD server, and Apache AXIS framework for Web services. They studied the percentage of different bug types, and their relationship with features including time to bug fix, severity and bug types.

What's more, some related studies are also performed. Chandra et al. [20] classified bugs into environment-independent bugs and environment-dependent bugs and examined the faults in three open source applications: Apache web server, GNOME, and MySQL. Authors in [21] defined a comprehensive bug manifestation characteristics and classified bugs according to their triggers, thus they have workload- and state-dependent triggers, and user- and environment-dependent triggers. In addition, some studies focus on specific bugs such as concurrency bugs [22], ARBs [23].

In this paper, we perform an empirical study on Android operating system. The main advantages of this work over existing ones include:

(1) A new research object. Android operating system is an embedded operating system with limited memory and CPU. Due to its popularity, the requirement of its performance and reliability is much more demanding. It becomes urgent for developers to improve user experience based on the feedbacks.

(2) Several attributes related with bug location, fixing and user behaviors are considered in our study. The relationships between these attributes and bug types are explored.

(3) A comparison between Linux and Android in terms of bug proportions is conducted, which provides a new perspective to understand these two operating systems.

To the best of our knowledge, this is the first work in Android operating system to perform bug classifications with fault triggers.

III. Bug Source And Classification Procedure

A. Bug Classification Approach

Due to the focus of this work, the bug classification criteria proposed in [12] are used here. According to the complexity of fault activation and error propagation, bugs are classified as Bohrbugs and Mandelbugs [10]. Mandelbugs are further classified as aging related bug (ARB) and non-aging related Mandelbugs (NAM). The subtypes of NAMs are:

- TIM: the timing of inputs and operations is the factor that affects fault activation and/or error propagation;
- SEQ: the sequencing of inputs and operations is the factor that influences the activation and/or error propagation (the inputs could have been run in a different order, and at least one of the other orders would not have resulted in a failure);
- ENV: the interaction of software application with its system-internal environment is the factor that impacts faults activation and/or error propagation;
- LAG: a time delay exists between the activation of bug and the occurrence of failure;
- NAU: Sometimes, insufficient information is provided about fault activation and error propagation conditions. However, in some reports, it is mentioned that failures can be reproduced occasionally. Therefore, these bugs are classified as a NAM of unknown subtype (NAU).

The subtypes of ARBs are shown as follows:

- MEM: a kind of ARB related to the accumulation of errors resulted from improper memory management (e.g., memory leaks, buffers not being flushed);
- STO: a kind of ARB related to the accumulation of errors caused by improper storage space management (e.g., disk space is consumed by the bug);
- LOG: a kind of ARB resulting in leaks of other logical resources (system-dependent data structures e.g., inodes or sockets that are not freed after usage);
- NUM: a kind of ARB which leads to accumulation of numerical errors (e.g., integer overflows, round-off errors);
- TOT: The fault activation or error propagation rate increases with total system runtime, but it is not induced by accumulation of internal error states.
- ARU: In some cases, the information about failure mechanism (e.g., the presence of error accumulation) is insufficient, so it is difficult to decide which subtype this bug should belong to. However, some reports mention that the failure rate tends to increase over time and the occurrence of failure will undergo certain accumulation process. In this case, the bug is classified as an unknown type of ARB.

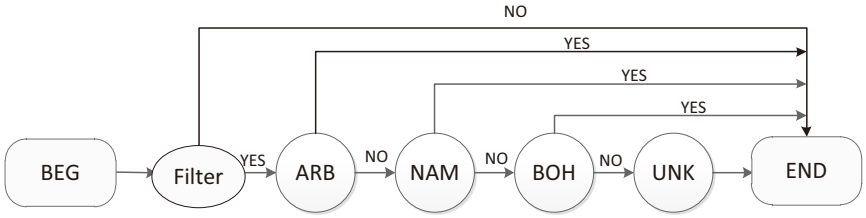

Fig. 3. Bug classification flow chart

B. Bug Source

Cyanogenmod is a widely used Android operating system [14]. With millions of users on dozens of devices, a public and actively used bug repository called JIRA[1] is set up for Cyanogenmod. In JIRA, all the reports about bugs, feature ideas, and other aspects of the project are tracked and organized together. Furthermore, the attributes of each bug is given, such as type (bug, new feature etc.), status (open, closed, etc.), priority, resolution (fixed, duplicate, etc.), affects versions. Reports in JIRA can be filtered according to different requirements. In order to have a comprehensive analysis of this operating system, all bugs in Cyanogenmod bug report are the possible candidates for our analysis. Furthermore, bugs are picked up in terms of following three conditions:

(1) Bugs which have been fixed and closed. This kind of reports can provide sufficient and reliable information.

(2) Bugs on released versions. Released versions are more stable than unreleased ones.

(3) Bugs reported from January 29th, 2013 to August 31th, 2015. The former date is the day that the first bug in Cyanogenmod is reported in this repository we can track now, and the latter date is the day we gathered the empirical data.

According to the criteria above, 513 bugs are selected for our study among more than 7000 reports related with Cyanogenmod.

C. Bug Classification Procedure

The classification process is based on the description, comments, the fixing code and attachments in the bug report. The classification procedure is conducted manually as shown in Fig. 3.

1) Given a bug report, the first step is to determine whether it is definitely a bug in Cyanogenmod, and whether it belongs to three conditions in Section III.B. The bug reports do not meets conditions are excluded from our analysis. For example, bugs in nightly version (e.g., CYAN-5132), bugs in self-installed APP (e.g., CYAN-5082), bug reports that are closed because related functions are not provided in the updated version (e.g., CYAN-3707).
2) Then we will consider whether it can be classified as an ARB according to the characteristics of each subtype: MEM, STO, LOG, NUM, TOL and ARU.
3) If not, we will try to figure out whether it belongs to NAM. The subtypes include TIM, ENV, SEQ, LAG and NAU.
4) If not, and the bug activation and error propagation are simple, we will classify the bug as a BOH.
5) If there is not enough information to classify bugs, it will be classified as UNK.

The classification is performed independently with three of the authors and doubtful cases are discussed to make a consensus. According to the first step in this procedure, we have ruled out 36 bugs. Thus there are totally 477 bugs left in our empirical study. To make our classification method clearer, some examples are shown in Table II. Furthermore, we release our data to our research website[2], enabling other researchers perform and understand the classification more easily.

During the classification process, several bug attributes are also recorded: component, priority, created and resolved time.

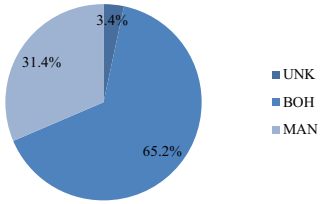

Fig. 4. Proportion of bugs

IV. Empirical Result

This section provides the result of bug classification in Android. We analyze the result from three dimensions: bug types, bug attributes, comparison analysis between Android and Linux.

A. Bug Type of Android

Different bug types usually need different detection techniques and countermeasures. In Fig. 4, we classify the examined bugs into three types: Bohrbugs, Mandelbugs and Uknown.

> Finding (1): About two thirds (65.2%) of the examined bugs are Bohrbugs and the fraction of Mandelbugs is 31.4%.

From Finding (1), we can observe that, although unit testing, functional testing and integration testing have been conducted before the release of Android operating system, a large number

[1]https://jira.cyanogenmod.org

[2]http://zhengzheng.buaa.edu.cn/en/pdf/data.xlsx

TABLE II
SOME EXAMPLES OF BUGS AND THEIR CLASSIFICATION STEPS

Bug ID	Type	Description	Bug Classification Procedures
CYAN-3746	MEM	In Nexus 5, the operations of swiping home screens or going to apps from home screen are usually choppy. Typically when the scoll ends, it is still visible in the last moment in most cases. A memory leak occurs in the system.	Because memory leak is detected in the system, which is related to improper memory management, so this bug is classified as MEM.
CYAN-504	ARU	The steps to trigger bug are as followed: 1) Turn on Bluetooth under settings. 2) Connect the device to be paired. 3) Turn off Bluetooth. Repeat these steps for a few times and the failure might surface. The Bluetooth shows Bluetooth is turned off, but actually not.	The failure surfaces after execution of step 1-3 for a few times. It is related to certain accumulation process, but neither memory management, storage space, accumulation of numerical errors, nor TOT. Thus this bug is classified as ARU.
CYAN-2380	SEQ	If Bluetooth of cell phone is turned on before we start the car, it works well. But when we change the order of these two operations, there will be something wrong with the connection. In this situation, the connection lasts only for a few minutes and then BT connection failed will be shown on the screen.	At first, the bug activation or error propagation isnt related to the total time that a system has been running, so it is not an ARB. Secondly, from the description, the failure is related with the sequence of starting car and openning Bluetooth. Different sequence matters in terms with the surface of failure.
CYAN-5583	ENV	The reproduction steps are: 1. open camera and set flash to ON; 2. take one picture; 3. set flash to OFF; 4. take another picture. After these four steps, the camera application crashes.	The failure occurs after four operations, which constitutes the complex environment for the failure to surface.
CYAN-4823	LAG	After Netflix playback runs for about 30 seconds, there will be green flickering on the screen in the playback areas.	There is time delay between the bug activation and failure occurrence, so it is classified as LAG.
CYAN-4892	NAU	On home screen, every now and then there will be a random extra and empty page made by Trebuchet without adding it manually.	No indications that shows bug activation or error propagation rate grows with the total running time of the system. However the failure occurs randomly, and it is difficult to reproduce. Thus this bug is classified as NAU.
CYAN-2606	BOH	Whenever user tries to switch to GSM/WCDMA auto, com.android.phone crashes.	Both the bug activation and error propagation are not complex, so it is classified as BOH.

of Bohrbugs still remain. This may attribute to the great quantity of Bohrbugs introduced during the development process. Even they are easier to be isolated and a great number of Borhbugs have already been fixed during testing, the number of Bohrbugs in the released versions is still large, leading to the high proportion of Bohrbugs in the released versions.

Furthermore, although the number of Mandelbugs in Android is less than Bohrbugs, it constitutes a non-negligible part. The share of Mandelbugs (31.4%) is close to that in previous studies, such as 38% in MySQL [12], 36.5% in a space mission system software [13], and 20-40% in [24]. While software testing is effective against Bohrbugs which are easily isolated and reproduced, it is considerably less suitable for tackling with Mandelbugs due to its non-deterministic nature.

Implications: *As to Bohrbugs, the previous testing is insufficient, more unit testing, function testing, as well as integration testing are necessary. As to Mandelbugs, a non-negligible fraction exists. Due to their difficulty in fixing, cost-effective fault tolerance techniques, such as environment diversity [25], should be developed to handle them at run time.*

To better understand the characteristics of Mandelbug, we further analyze the bug proportions of each subtype in NAM and ARB respectively in detail.

Finding (2): almost all (97.6%) of the examined NAMs belong to three subtypes: ENV, LAG and NAU. Half of NAMs are of subtype ENV.

The proportion of subtypes in NAM is shown in Fig. 5. Examples of these three subtypes are:

- ENV: in CYAN-4054, the following operations are conducted: a. open camera and take one picture; b. change to video mode and set the FPS to 90 in Settings; c. return to home screen by pressing home button; d. click camera icon. After these four steps, the camera application will crash immediately.
- LAG: in CYAN-4823, green flickering on the screen in the playback areas may appear after Netflix playback has been running for about 30 seconds.
- NAU: in CYAN-4892, a random extra and empty page made by Trebuchet may appear on home screen automatically every now and then.

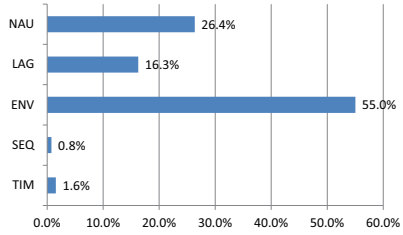

Fig. 5. Proportion of NAM sutbypes

Obviously, problems with similar subtypes occur frequently in our daily cellphone usage. Finding (2) confirms user experience.

ENV takes the biggest share of NAM. In the past, many recent studies specifically focus on environment dependent bugs [20], [21], [26]. In Android, operations are usually conducted under complex configurations. Meanwhile, the switch among different activities also makes applications environment complicated (e.g., CYAN-4054). Moreover, the widely and

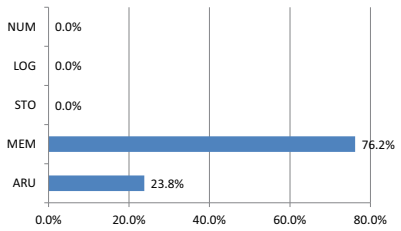

Fig. 6. Proportion of ARB sutbypes

frequently utilization of Bluetooth, network function, headphone function, which are usually used together with other applications or components, gives rise to the chance the ENV occurs.

As for LAG, a time delay exists between the activation of a bug and the occurrence of failure. An example is CYAN-4823. A user uses the playback function of Netflix to play a video. There is no failure at the beginning of the execution. After about 30 seconds, although Netflix plays the same video, a failure occurs.

The percentage of NAU is 26.4%, which accounts for a non-negligible part in Mandelbugs. For this kind of bugs, although the exact bug manifestation processes are not presented in their bug reports, we classify them into Mandelbugs according to the described phenomenon in the reports that they cause failures sporadically under the same condition (e.g., CYAN-4892).

Implications: *To reduce the impact of Mandelbugs in Android, bug detection, testing or fault tolerance should focus on these three major bug subtypes, especially ENV. For ENV, testing under multi-configurations and frequent function switch in an application, and two or more applications run together should be paid attention to.*

Finding (3): Most (76.2%) of the examined ARBs are of MEM subtype.

The proportion distribution of subtypes of ARBs is shown in Fig. 6. Finding (3) is similar with the observations in previous studies in other software systems that most of ARBs are related to memory management [12], [13], [27]. Moreover, during the bug classification procedure, we find that memory leak is the primary memory management problem.

The main part of Android application is programed with Java language. Although Java has its own memory management mechanism (for example, garbage collection), it is not adequate to ensure the proper memory management. Many applications suffer memory leak. In Android, memory management bugs are common due to the following reasons. Firstly, Android framework transfers some duties of memory management to the developers, while developers may assume that the framework will manage the memory release automatically so they might elide a call to release a resource. Secondly, the functionalities of operating system are put in the first place by developers, and performance testing is usually not performed adequately before release. At last, even an experienced developer may not release all the resources along all possible sequences of event handlers [28].

TABLE III
COMPONENT LIST

Audio, Bluetooth, Browser, Busybox, c-apps, Calendar, Camera, Clock, Contacts, File Manager, FM Radio, Framework, Gallery, GPS, Graphics, I18n, Kernel, Launcher, Lights, Lockscreen, Mail, Media Scanner, MobileData, Music, NFC, Phone, Privacy Guard, Profiles, Quiet hours, Recovery, RIL, Settings, SMS/MMS, Superuser, Themes, Translations, UI, Updater, USB Tether, Utilities, VideoPlayback, VPN, Weather, Wifi, Wifi Tether, Wimax

TABLE IV
MEAN AND STANDARD DEVIATION OF #BUGS

	Mean	Std.dev
Bohrbug	18.31	5.51
Mandelbug	8.62	6.16

Implications: *The memory characteristic in Android has been attracting large attention [28]–[32]. Developers are suggested to pay special attention to the resource release. Moreover, more performance testing is required to reduce memory leaks.*

B. Bug Attributes Analysis in Android

In this section, we will explore the relationship between bug attributes and bug types in Android from following three aspects.

1) Bug types vs. components

In this subsection, we will investigate the bug type distribution among different components. The components of Cyanogenmod in JIRA bug repository are listed in Table III.

In this study, the bug reports without component assigned are excluded and we get 396 bugs finally. Note that for some bugs, there could be more than one component marked, so the sum of bugs over all the components are greater than the total number of bugs analyzed.

From Table III, it is clear that there are 47 components in total. Since there is little significance in studying components with a small number of bugs, components with no more than 10 bugs are excluded from our study. Thus 13 components with 268 bugs are studied here. These 13 components and their corresponding number of Bohrbugs and Mandelbugs are shown in Fig. 7.

Finding (4.1): Bohrbugs are almost uniformly distributed in components, while Mandelbugs are prone to lie in the components of Camera, Audio and Bluetooth.

The mean and standard deviation of the number of two types of bugs among these 13 components are list in Table IV. From the table, we can see that the standard deviation of Mandelbugs across components is 6.16, while the value of Bohrbugs is 5.51. The low value of standard deviation implies less fluctuation of bug numbers across components. Thus comparing with Bohrbugs, Mandelbugs tend to locate in a few components. The number of Mandelbugs in top-3 components constitutes around 50% of that in 13 components studied here.

What's more, it can be obviously observed from Fig. 7 that in most components there are more Bohrbugs than Mandel-

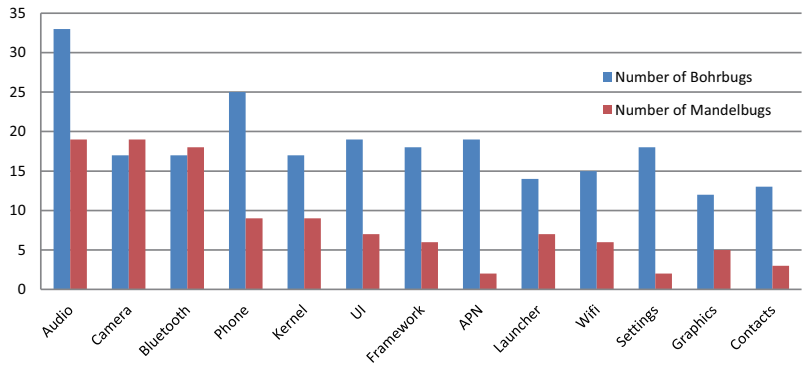

Fig. 7. #Bug in 13 components

bugs. This phenomenon is another support view to the large number of Bohrbugs in finding (1).

Finding (4.2): Camera and Bluetooth seem to have the same number of Bohrbugs and Mandelbugs.

The phenomenon of Finding (4.2) may be related with the functionalities of these two components. Camera is a component responsible for capturing pictures or videos. ENV accounts for the largest share of Mandelbugs in camera component. Specifically, 12 of 19 Mandelbugs are ENVs. The frequent switch between camera mode and video mode, as well as some settings along with these two modes construct complex environment for the bug manifestation. For example, the reproduction of CYAN-5583 is as follows: 1. open camera and set flash to ON; 2. take one picture; 3. set flash to OFF; 4. take another picture. After these four steps, the camera application crashes.

Bluetooth is widely used in mobile phones to exchange data over short distances, as well as build personal area networks [33]. The large number of Mandelbugs in Bluetooth component is consistent with our intuitions. Bluetooth usually involves two devices, and it is commonly used in conjunction with other applications (such as phone calls, music applications). Furthermore, other operations can be conducted simultaneously when data are transferred by Bluetooth, such as typing the keyboard while listening to music with Bluetooth headset. All the cases above show that bugs in Bluetooth are highly related to interaction with system environment. From the classification results, we can see that most of the Mandelbugs in Bluetooth component belong to the subtype ENV. In detail, 11 bugs are classified as ENV among 18 Mandelbugs in total.

As to the components of Camera and Bluetooth, considering from functionalities of these two components, the great number of Mandelbugs can reach the same scale of Bohrbugs they suffer.

Implications: *For detecting Mandelbugs, components Camera, Audio and Bluetooth are suggested to be carefully examined. Meanwhile, components Camera and Bluetooth should be noticed for their large percentage of Mandelbugs.*

2) Bug types vs. fix duraiton

In this section, we will explore whether the fix duration of bugs is related to bug types. It is intuitive that the bugs with more complex manifestation process need longer time to be fixed. Because there are no exact data of fixing time provided in bug reports, in our research we use the time span between the open time and the resolved time of a bug report as an approximation of its fix duration in terms of days.

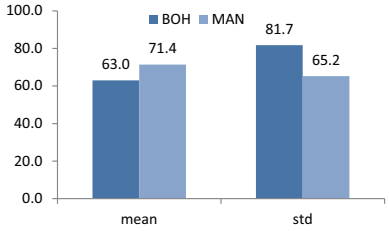

Fig. 8. The relationship between bug type and fix duration

Finding (5): In Android, it tends to take more time to fix Mandelbugs than Bohrbugs.

The mean and standard deviation of two bug types are shown in Fig. 8. From the figure, we can observe that the average time to fix a Borhnbug is 63.0 days. Comparatively, the average time to fix a Mandelbug is 71.4 days. This phenomenon is consistent with what we suppose originally. In addition, we further apply nonparametric hypothesis test, Mann-Whitney-Wilcoxon Test [34], to verify the result. The null hypothesis is that the fix duration of Bohrbugs and Mandelbugs are sampled from the same distribution. After performing the test, we got that the p-value is 0.009. Therefore it is obvious that the null hypothesis of no significant difference at 95% of confidence is rejected. Thus we can get the conclusion that Mandelbugs tend to require more time to be fixed. This also provides evidence of the relationship between bug types and their fix duration in previous studies in Linux, HTTPD, and AXIS [12].

This phenomenon can be explained by the difference between bug trigger conditions and bug properties. The first thing a developer needs to do when he is assigned a bug is reproduce the failure. For a Mandelbug, due to its complexity, the original information a reporter provides may be insufficient to identify the bug activation and the way to reproduce the failure. Furthermore, with the nondeterministic property of

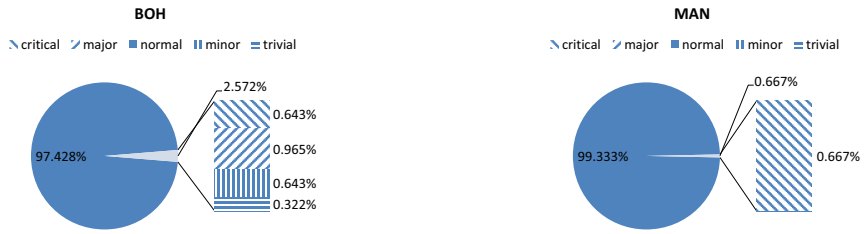

Fig. 9. The relationship between bug priority and bug type

Mandelbugs, the right failure reproduce procedure may not always make the failure occur. As a consequence, it may take longer to fix a Mandelbug.

Implications: *For Mandelbugs, due to their long fix duration, other mitigation techniques (such as retry, restart, reboot) can be utilized to prevent their manifestation.*

3) Bug types vs. priority

Bug priority indicates the relatively importance of a bug compared to other bugs [35]. In the bug tracking system JIRA, five priorities are used to denote the importance: critical, major, normal, minor, and trivial. We try to explore whether there is certain correlation between bug types and their priority in this section.

Finding (6): In Android, the bug priority is not influenced by bug type.

The priority distribution in each type of bug is listed in Fig. 9. Moreover, the nonparametric statistics method Pearson chi-square test is adopted, assessing the null hypothesis that two types of bugs are independent [34]. The confidence is set to 95%. From the test, we got p-value 0.569. Thus the null hypothesis could be accepted.

Fig. 9 shows that most of Bohrbugs and Mandelbugs belong to normal priority. It can be observed that the complexity of bug activation conditions and error propagation do not have much relationship with the significance of bugs users perceived. From users' perspective, bugs correlated with emergency situation (e.g., CYAN-3642: calling 911 doesnt work) or internet security (e.g., CYAN-3502: the operating system is vulnerable to MITM when the updater utilizes HTTP to check for updates) has the highest priority, since these are the situations having more relationship with user interests.

Implications: *For mitigating Bohrbugs and Mandelbugs during design, the relative importance of these two kinds of bugs is prone to be related with their proportions.*

C. Comparison Analysis Between Android and Linux

Linux and Android are two most popular operating systems. Although Android is developed based on the Linux kernel, it is designed primarily for touchscreen mobile devices. Android has further architectural changes outside the typical Linux kernel development cycle, such as the inclusion of Binder, logger, anonymous shared memory driver (ashmem), different out-of-memory handing [36]. Furthermore, more and more researchers have paid attention to the comparison between Android and Linux [37], [38]. In this section, a bug proportion

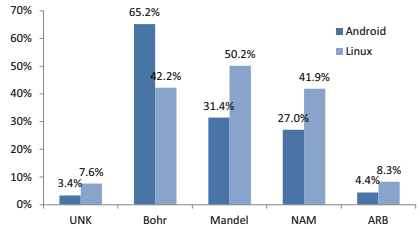

Fig. 10. Comparison of bug subtype proportions in Android and Linux

comparison between Android and Linux is conducted. Fig. 10 illustrates the comparison of the bug distribution in Linux and Android. The data of Linux are extracted from [12].

Finding (7): Although Android is developed based on Linux, their bug percentages are quite different.

From Fig. 10, we can see that the bug distributions of Linux and Android are different from each other in terms of every type of bugs. For example, Mandelbugs share about 31.4% in Android, while in Linux the percentage is 50.2%.

There are several reasons that might account for this phenomenon. Firstly, due to its size and portable trait, the number of equipments that can connect with Android is usually much less than that in Linux. Thus in general Android has less prone to interact with hardware than Linux. Secondly, some new mechanisms in Android (e.g., low memory killer) may reduce the probability of failures caused by Mandelbugs.

Finding (8): The percentage of ARBs in Android is almost half of that in Linux.

Specifically, it is depicted in Fig. 10 that the percentage of ARB is 4.4% and 8.3% in Android and Linux, respectively. From previous studies, MEM is the primary subtype of ARBs [12], [13], [27]. So from the perspective of MEM, there are two reasons accounting for this phenomenon. First of all, in the aspect of programming language, applications in Android (each application in Android runs in its own Dalvik virtual machine) are mainly programmed by Java which has garbage collection mechanism to help developers take memory management within the Dalvik virtual machine, leading to the low percentage of bugs related to MEM. While C/C++ is used in Linux, developers should manage dynamic memory allocation and deallocation by themselves.

Secondly, the memory management is different. The Low Memory Killer (LMK), an enhancement of the Out Of Memory killer (OOM killer) in the original Linux Kernel, is

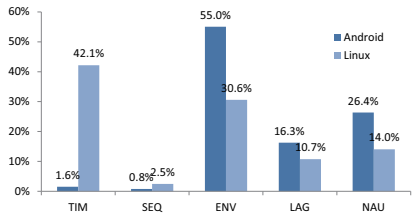

Fig. 11. Comparison of bug subtype proportions in NAM

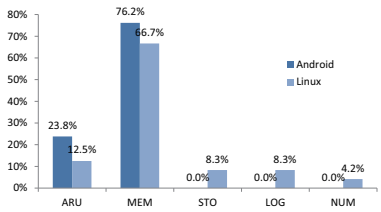

Fig. 12. Comparison of bug subtype proportions in ARB

developed to manage memory in Android. Process in android is divided into five levels in the importance hierarchy [39]: foreground process, visible process, service process, background process, and empty process, in the priority decreasing order. If the system is in low memory condition and certain threshold is reached, LMK will first kill empty process to free memory. If memory is still not enough, background process will be killed. Foreground process is the last group of process that will be killed. While in Linux, the kernel Out Of Memory killer (OOM killer) takes advantage of heuristics and computes the processs badness in order to determine which process will be killed in memory pressure situation [40]. Thus, if memory leak in Android happened in the process with low priority, the operating system will reclaim the memory if needed without the influence of the foreground process.

Implications: *Although Android is developed based on the Linux kernel, developers could not use the benefits in Linux directly in Android. For mitigating ARBs in Android, Developers can design multi-level software rejuvenation strategies (such as close the application from recent app and restart it, reboot the phone) on the basis of the memory usage to get better user experience.*

Finding (9): Among the subtypes of NAM, the percentage of TIM bugs in Android is significantly less than those in Linux.

In Fig. 11, the proportions of NAM subtypes in Android and Linux are provided. From the figure, we can see that in the subtypes of ENV, LAG, and NAU, Android has a higher proportion than Linux. While in the subtype of TIM, Linux occupies higher proportion.

Among above phenomena, a noticeable one is the difference in TIM. From the figure, it can be observed that the percentage of TIM in Linux is 42.1%, while in Android, it is 1.6%. Although Android is a multiuser operating system (each application is regarded as a user in Android, and usually every application has its own process), due to its limited resources and screen size, only a limited number of foreground processes are allowed to run at the same time (usually one to a few). Even background or service processes can be executed in the meantime, but their functionalities would be limited comparing with foreground processes, such as receiving push notifications [32]. For example, when a mail application is running as foreground process, functions such as sending emails is able to be performed. While when it is running as service process, only some services such as email notification are available. Thus the number of concurrent processes running on Android is small, leading to the less possibility of TIM bugs.

Implications: *Developers dont need to pay too much attention to TIM subtype in Android as that in Linux.*

Finding (10): MEM subtype is dominant among ARBs in both Android and Linux.

The comparison of the proportions of ARB subtypes in Android and Linux is summarized in Fig. 12. From the figure, it is obvious that there are not apparent proportion differences in these bug subtypes between Android and Linux.

The MEM proportion in ARB is 76.2% and 66.7% respectively in Android and Linux respectively. Irrespective of different mechanism and design purpose of two operating system, MEM constitutes the greatest part of ARB, which supports the conclusion of previous studies [12], [13], [27].

Implications: *Refer to implications corresponding to Finding (8).*

V. Threats to Validity

Similar to other empirical studies, our study is also subject to validity problem. Potential threats to the validity of our study include bug report selection, bug report description, and manual inspection.

In terms of bug report selection, we only select fixed and closed issues, since unfixed and unclosed issues may contain incomplete information. Bug type proportions and results for bug attributes may be different if unfixed and unclosed issues are included.

In terms of bug report description, since the reports are written by users and developers, the accuracy and completeness of the description and comments influence the judgement of bug types. Furthermore, the component and bug priority of each bug report are extracted directly from the issue tracking system. Due to the reporters' different understanding, there might be bias about it.

In terms of manual inspection, although we have inspected carefully all the related information in bug reports, including descriptions, comments, as well as attached files and patches, the possible classification mistakes could not be avoided.

VI. Conclusion

In this paper, we performed an empirical investigation of bugs in Android operating system in terms of fault triggers. With the bug classification results based on 513 real world bug reports, our analyses are conducted from three dimensions: the bug types of Android, bug attributes analysis in Android, and bug type comparison between Android and Linux, along with some interesting findings and implications that can be

adopted by developers or users. Future research on Android can benefit from our study. For example, bug detection or fault tolerance strategies can mainly focus on ENV subtype to reduce the impact of Mandelbugs. As to the components of Android, Camera, Bluetooth and Audio should be firstly examined to reduce nondeterministic behaviors.

ACKNOWLEDGMENT

Zheng's research was partially supported by Aeronautical Science Fund of China under grant number 20155551025 and by State Key Laboratory of Software Development Environment under grant number SKLSDE-2013ZX-08. Trivedi's research was partially supported by US Navy NEEC under grant number N00174-16-C-0036 and by US NSF under grant number CNS-1523994.

REFERENCES

[1] Android, the world's most popular mobile platform. [Online]. Available: http://developer.android.com/about/android.html

[2] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 1457–1462.

[3] H. Shahriar, S. North, and E. Mawangi, "Testing of memory leak in android applications," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*. IEEE, 2014, pp. 176–183.

[4] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications." in *ICSME*, 2014, pp. 121–130.

[5] A. K. Maji, K. Hao, S. Sultana, and S. Bagchi, "Characterizing failures in mobile oses: A case study with android and symbian," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 249–258.

[6] N. Klein, C. S. Corley, and N. A. Kraft, "New features for duplicate bug detection," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 324–327.

[7] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 183–192.

[8] M. Asaduzzaman, M. C. Bullock, C. K. Roy, and K. A. Schneider, "Bug introducing changes: A case study with android," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 2012, pp. 116–119.

[9] J. Gray, "Why do computers stop and what can be done about it?" in *Symposium on reliability in distributed software and database systems*. Los Angeles, CA, USA, 1986, pp. 3–12.

[10] M. Grottke and K. Trivedi, "Software faults, software aging and software rejuvenation," *Journal of the Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.

[11] M. Grottke and K. S. Trivedi, "Fighting bugs: Remove, retry, replicate, and rejuvenate," *Computer*, vol. 40, no. 2, pp. 107–109, 2007.

[12] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi, "Fault triggers in open-source software: An experience report," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2013, pp. 178–187.

[13] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *2010 IEEE/IFIP international conference on dependable systems & networks (DSN)*. IEEE, 2010, pp. 447–456.

[14] Wikipedia, cyanogenmod. [Online]. Available: https://en.wikipedia.org/wiki/CyanogenMod

[15] N. Mellegård, M. Staron, and F. Törner, "A light-weight defect classification scheme for embedded automotive software and its initial evaluation," in *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. IEEE, 2012, pp. 261–270.

[16] R. B. Grady, *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[17] "Ieee std. 1044-2009. standard classification for software anomalies." IEEE, 2010.

[18] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[19] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *IEEE Proceedings of Workshop on Software Aging and Rejuvenation, in conjunction with ISSRE. Seattle, WA*, 2008.

[20] S. Chandra and P. M. Chen, "Whither generic recovery from application faults? a fault study using open-source software," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*. IEEE, 2000, pp. 97–106.

[21] D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, "How do bugs surface? a comprehensive study on the characteristics of software bugs manifestation," *Journal of Systems and Software*, vol. 113, pp. 27–43, 2016.

[22] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," in *ACM Sigplan Notices*, vol. 43, no. 3. ACM, 2008, pp. 329–339.

[23] F. Machida, J. Xiang, K. Tadano, and Y. Maeno, "Aging-related bugs in cloud computing software," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 287–292.

[24] R. Chillarege, "Understanding bohr-mandel bugs through odc triggers and a case study with empirical estimations of their field proportion," in *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on*. IEEE, 2011, pp. 7–13.

[25] K. S. Trivedi, M. Grottke, and E. Andrade, "Software fault mitigation and availability assurance techniques," *International Journal of System Assurance Engineering and Management*, vol. 1, no. 4, pp. 340–350, 2010.

[26] D. G. Cavezza, R. Pietrantuono, J. Alonso, S. Russo, and K. S. Trivedi, "Reproducibility of environment-dependent software failures: An experience report," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 267–276.

[27] D. Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics," *Performance Evaluation*, vol. 70, no. 3, pp. 163–178, 2013.

[28] C. Guo, J. Zhang, J. Yan, Z. Zhang, and Y. Zhang, "Characterizing and detecting resource leaks in android applications," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 389–398.

[29] P. Stirparo, I. Nai Fovino, and I. Kounelis, "Data-in-use leakages from android memory-test and analysis," in *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 7-9 Oct. 2013, Lyon, France*. IEEE, 2013, pp. 701–708.

[30] R. S. Vairagde, R. A. Kulkarni *et al.*, "Mobile device oriented image scaling for reducing memory consumption in storing in android," *International Journal of Computer Applications Technology and Research*, vol. 3, no. 1, pp. 84–87, 2014.

[31] J. Park and B. Choi, "Automated memory leakage detection in android based systems," *International Journal of Control and Automation*, vol. 5, no. 2, pp. 35–42, 2012.

[32] S.-H. Kim, S. Kwon, J.-S. Kim, and J. Jeong, "Controlling physical memory fragmentation in mobile systems," *ACM SIGPLAN Notices*, vol. 50, no. 11, pp. 1–14, 2016.

[33] Wikipedia, bluetooth. [Online]. Available: https://en.wikipedia.org/wiki/Bluetooth

[34] D. J. Sheskin, *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[35] Cyanogenmod. [Online]. Available: http://www.cyanogenmod.org/

[36] Wikipedia, android (operating system). [Online]. Available: https://en.wikipedia.org/Android_(operating_system)

[37] F. Khomh, H. Yuan, and Y. Zou, "Adapting linux for mobile platforms: An empirical study of android," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 629–632.

[38] F. Maker and Y.-H. Chan, "A survey on android vs. linux," *University of California*, pp. 1–10, 2009.

[39] Android Developers, processes and Threads. [Online]. Available: https://developer.android.com/guide/components/processes-and-threads.html

[40] S. Brahler, "Analysis of the android architecture," *Karlsruhe institute for technology*, vol. 7, p. 8, 2010.