Experience Report: Fault Triggers in Linux Operating System: From Evolution Perspective

Guanping Xiao*, Zheng Zheng*‡, Beibei Yin*, Kishor S. Trivedi†, Xiaoting Du*, Kaiyuan Cai*
*School of Automation Science and Electrical Engineering, Beihang University, Beijing, China
†Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
Email: {gpxiao, zhengz, yinbeibei, xiaoting_2015, kycai}@buaa.edu.cn, ktrivedi@duke.edu

***Abstract*—Linux operating system is a complex system that is prone to suffer failures during usage, and increases difficulties of fixing bugs. Different testing strategies and fault mitigation methods can be developed and applied based on different types of bugs, which leads to the necessity to have a deep understanding of the nature of bugs in Linux. In this paper, an empirical study is carried out on 5741 bug reports of Linux kernel from an evolution perspective. A bug classification is conducted based on fault triggering conditions, followed by the analysis of the evolution of bug type proportions over versions and time, together with their comparisons across versions, products and regression bugs. Moreover, the relationship between bug type proportions and clustering coefficient, as well as the relation between bug types and time to fix are presented. This paper reveals 13 interesting findings based on the empirical results and further provides guidance for developers and users based on these findings.**

***Index Terms*—bug classification; fault trigger; Linux; evolution; Mandelbug; regression bug;**

I. Introduction

The dependence on services provided by software systems keeps increasing. Consequently, failures in software systems have a large impact on our daily life. As a result, it is crucial to ensure the reliability of these software systems. Since an operating system is a special software system which interacts with hardware devices and provides operating environments to the software executing on a computer, its reliability has a direct influence on the reliability of the running software systems and the services they provide.

Obviously, it is not cost-effective to ensure high reliability of an operating system through exhaustive testing. Thus, failures will inevitably manifest after the system is deployed. However, these failures should be resolved as soon as possible to reduce service outage time. It can be expected that understanding the original source of faulty code and the factors that trigger faults and/or propagate errors, could provide valuable insights into software development and maintenance phases [1]. Therefore, it is necessary to explore bug data of an operating system.

Among various operating systems, Linux operating system is typical and well deployed in most fields of the society. With the development of Linux, an enormous amount of bug data has been accumulated and can be obtained publicly for analysis. Previous studies have concentrated on Linux bug data from several aspects, such as, patch analysis [2], [3], bug characteristics analysis [4], bug reworking analysis [5], predicting time to fix [6], etc. One interesting research topic is bug type classification and its characteristic analysis [7], [8], [9].

Previous researchers have classified bugs according to the failure manifestation perspective, such as hard and soft faults [10]. For comprehensively investigating fault triggers, which refer to the set of conditions that activate a fault and propagate the resulting error(s) into a failure, Grottke and Trivedi [11], [12] proposed the terminology Mandelbug (MAN) as the complementary antonym of Bohrbug (BOH). Unlike Bohrbug, that is a bug which can be easily isolated and reproduced, the activation and/or error propagation of a Mandelbug are complex. Moreover, Mandelbugs have two subtypes: non-aging related Mandelbug (NAM) and aging related bug (ARB). Aging related bugs are a type of bugs that can cause an increasing failure rate and/or degraded performance, known as software aging [13]. According to the above classification, researchers in [8] extended a more detailed bug type classification based on the different kinds of complexity in fault triggering conditions and analyzed bugs in four large open-source software systems including Linux, MySQL, HTTPD and AXIS.

Our study is performed with 5741 bug reports of Linux kernel. This represents a significant extension of the work in [8], whose data set of Linux kernel was 346 bug reports, focusing on four products/components and version 2.6 series. Moreover, a further study of bug type proportions from an evolution perspective involving versions, products, regressions and software metric aspects, and so forth, is performed. For each report, we examine the bug description, comments, as well as attached files carefully. In this study, we investigate the following four research questions.

RQ1: How do proportions of bug types in Linux evolve over versions or time?

Linux has put out more than 1300 releases ranging from versions 1.0 to 4.1 over the past 20 years [14]. What the bug type proportions in Linux are and how they evolve over versions or time, is to be explored.

RQ2: How does the proportion of regression bugs in Linux evolve over versions or time?

With the evolution of Linux, maintaining becomes increasingly difficult [15], and several problems would be occurred, such as regressions. A Linux regression is a bug that exists in some versions of Linux and did not exist in a previous version

‡Corresponding author: Z. Zheng (Email: zhengz@buaa.edu.cn).

2332-6549/17 $31.00 © 2017 IEEE
DOI 10.1109/ISSRE.2017.21

TABLE I
SUMMARY OF FINDINGS OF BUG TYPES ANALYSIS IN LINUX.

Findings on bug types	
#1	Among actual bugs, 55.82% are BOHs and 36.34% are MANs.
#2	For NAMs, the major subtypes are TIM (37.23%), ENV (36.51%) and LAG (19.12%).
#3	For ARBs, more than two thirds (68.78%) are related to MEM.
#4	The proportion of BOHs tends to slowly increase with the evolution of versions or time, while the proportion of NAMs tends to slowly decrease. For ARBs, its proportion tends to slightly decrease over time. For all three types, the proportions stabilize around a constant value after 4000 days.
#5	The proportions of bug types and their evolution trends are different across versions. For all selected versions, the proportions of BOHs and MANs tend to stabilize around a constant value after 600 days.
#6	The number of bugs in products related to Drivers (Drivers and ACPI) accounts for 51.57% of all classified bugs.
#7	In products, a bug belonging to Drivers, ACPI or Platform is more likely a BOH; a bug occurring in File System, IO/Storage or Core (Memory, Process Management and Timers) is more likely an NAM or ARB; a Networking bug is more possible an NAM.
#8	Evolution trends of bug type proportions are different in products. The proportions of NAMs in products File System, IO/Storage and Core (Memory, Process Management and Timers) tend to slightly increase with time. The proportions of BOHs tend to slowly increase with time in all products. For ARBs, the proportions tend to stabilize around a constant value after about 3000 days.
Findings on regression bugs	
#9	More than half of the classified bugs are regression bugs.
#10	The proportion of BOHs in regression bugs is higher than that in non-regression bugs. Moreover, a regression bug is more likely a BOH, while a non-regression bug is more possible an NAM or ARB.
#11	The proportion of regression bugs tends to grow with the evolution of versions or time, and it stabilizes around a constant value 0.5 after 3500 days.
Findings on software metrics	
#12	With the evolution of clustering coefficient, Linux with a large clustering coefficient tends to have a low proportion of BOHs. In contrast, Linux with a large clustering coefficient tends to possess a high proportion of MANs.
Findings on time to fix	
#13	The average time taken to fix a MAN tends to be longer than that for fixing a BOH.

[16]. What the proportion of regression bugs in Linux is, and how it evolves over versions or time, as well as how it impacts the evolution of bug type proportions, will be answered in this research question. To the best of our knowledge, it is the first paper exploring the proportion of regression bugs in Linux.

RQ3: Is there a software metric that can reflect the evolution of bug type proportions?

What the relationship between bug type proportions and software metrics is, will be investigated in this research question.

RQ4: What's the relationship between bug types and time to fix?

In this research question, we examine the relationship between bug types and time to fix the bug.

Based on the empirical study on above four RQs, we list a set of findings summarized in TABLE I and their implications can be found in relevant parts in the following, which can provide useful guidance for developers and users.

The rest of the paper is organized as follows. Section II shows the related work, and Section III describes the data source, bug types, classification procedure, as well as a statistical metric used in this paper. In Section IV, results of our analysis are presented and discussed. Section V presents the threats to validity, and finally conclusion is given in Section VI.

II. RELATED WORK

There are many existing papers on defining general characteristics of defects, such as the IEEE Std. 1044 scheme [17], the Hewlett-Packard (HP) scheme [18] and the Orthogonal Defect Classification (ODC) [19]. ODC classifies defects through several attributes, in which the most important one is the defect type that captures the semantic of the fix made by the programmer, and the defect trigger. Meanwhile, ODC triggers are directly related to bug surfacing activities.

The classification method used in this paper is based on the bug manifestation dimension, or in other words, is based on the reproducibility of a bug. In 1985, Gray [10] introduced a systematic abstract about the reproducibility of a bug. He used solid or hard faults to refer to the easily reproducible failures (named Bohrbugs) and elusive or soft faults refer to the transient reproducible failures (named Heisenbugs). Grottke and Trivedi [11], [12] proposed a more general terminology of Mandelbugs: that are those bugs whose activation/propagation appears as non-deterministic and are complex, as the complementary antonym of Bohrbugs. It should be noted that, Mandelbugs are more general classification than Heisenbugs which is a subset of Mandelbugs. The definitions of Bohrbug and Mandelbug are as follows:

- **Bohrbug**: a kind of bug whose activation and error propagation are simple, thus it can be easily reproduced

and isolated, and its manifestation is consistent.

- **Mandelbug**: a kind of bug whose activation and error propagation are complex, thus it is hard to reproduce and its manifestation is transient, due to the possibility of the direct factor that a time lag between the fault activation and the failure occurrence, or the possible influence of indirect factor, such as system-internal environment, the timing of inputs and operations, and the sequencing of inputs and operations.

Moreover, Mandelbugs can be divided into aging related bugs and non-aging related Mandelbugs. Aging related bugs are a type of bugs that can lead to an increasing failure rate and/or degraded performance, causing software to appear to be aging [12]. The failure manifestation of an aging related bug is an accumulative process, related to memory management, storage, numerical errors, etc. [20]. The remaining Mandelbugs are defined as non-aging related Mandelbugs that do not cause software aging. In 2013, researchers in [8] proposed a more detailed subtype classification for non-aging related Mandelbugs and aging related bugs, based on the different kinds of complexity in fault triggering conditions. This provides more accurate information about bug types.

By adopting the above classification method, several researches have concentrated on bug reports classification and related analysis of different software systems. Grottke et al. [21] investigated the faults discovered in the on-board software for 18 JPL/NASA space missions. In their paper, among 520 software faults detected in all 18 missions, 61.4% were Bohrbugs and 36.5% were Mandelbugs. Researchers in [8] examined bugs in four large open-source software systems. They found that the proportion of Mandelbugs significantly varies among systems, i.e., the proportions of Mandelbugs for Linux, MySQL, HTTPD and AXIS are 50.2%, 38%, 17.5% and 7.5%, respectively. Moreover, Qin et al. [22] conducted a bug classification on Android operating system by exploring 513 bug reports and found that 31.4% of bugs in Android are Mandelbugs. Chandra et al. [23] examined the faults that occur in Apache web server, the GNOME desktop environment, and the MySQL database and found that 5–14% of the faults were triggered by transient conditions, such as timing and synchronization, that naturally fix themselves during recovery. Researchers in [24] investigated the characteristics of the bug manifestation process through defining a set of failure-exposing conditions, such as workload- and state-dependent triggers, user- and environment-dependent triggers. In addition, several studies focus on specific bugs, such as concurrency bugs [25], ARBs [26].

Another research needed to be discussed is regression bugs. A regression bug is a bug that causing a normal feature to stop working after a certain event (e.g., bug fixes, new feature work, etc.). One research paper concluded that regression bugs are very often caused by encompassed bug fixes included in patches [27]. Shihab et al. [28] conducted an industrial study on the risk of software changes and found that the number of bug reports and the developer experience are the best indicators of change risk. Khattar et al. [29] first performed an in-depth characterization study of regression bugs on Google Chromium project and they found that 51.09% of bugs in Google Chromium are regression bugs.

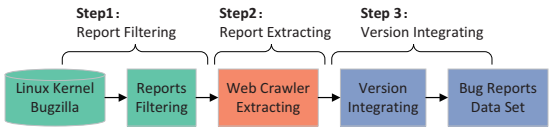

Fig. 1. Bug report collection and aggregation procedure. Step 1: report filtering; step 2: report extracting; step 3: version integrating.

In this paper, we perform an empirical study on Linux. The main advantages of our paper over existing ones include:

(1) A significant extensive data set. The total number of bug reports we analyzed is 5741, which is significantly larger than that of a previous study [8], not only extending the number of bugs, but also the numbers of products and versions.

(2) An evolution perspective is used to analyze bug type proportions in Linux, including evolution with versions and time. Furthermore, bug type proportions in different versions and products are also examined.

(3) Regression bugs in Linux and their bug types, as well as evolutions are examined. To the best of our knowledge, it is the first paper exploring the proportion of regression bugs in Linux.

(4) The relationship between bug type proportions and clustering coefficient is presented.

(5) The relationship between bug types and time to fix the bug is presented.

III. Approach

A. Data Collection and Aggregation

The bug data is collected from Linux kernel's official bug reporting website[1]. Fig. 1 depicts the data collection and aggregation procedure. Each step is described in detail in the following.

- **Step 1**: report filtering. Reports in Linux kernel Bugzilla were initially filtered based on conditions of "Status: CLOSED" and "Resolution: CODE_FIX".
- **Step 2**: report extracting. After filtering, the list of target reports was got and then each report was extracted to the local computer by a web crawler that we designed.
- **Step 3**: version integrating. The recorded versions are needed to be processed, due to the following two reasons. First, since some users used distribution versions that are based on Linux kernel, but they also reported problems in Linux kernel Bugzilla (e.g., recorded versions "Linux version 2.6.17 (Ubuntu 2.6.17-10.34-generic)", "2.6.15-rc4-686 (Debians -0experimental.1 build)", "2.6.12-gentoo-r6", etc.). Besides, some users compiled the latest source codes from Git (e.g., recorded versions "2.6.29-rc3-git20070311", "2.4.24rc3-git3", "2.6.25-rc1-git1", etc.),

[1]https://bugzilla.kernel.org/.

TABLE II
DETAILS OF DATA SET.

Versions	Products	Reports	Time frame
2.4-4.9	All	5741	Nov 2002 - Nov 2016

but not the formal release versions. According to the version numbering method of Linux kernel [14], [15], the recorded versions were integrated to major kernel versions. For instance, recoded version 2.6.11.7 is regarded as 2.6.11, since version 2.6.11 is a major version whereas version 2.6.11.7 is a minor version of 2.6.11.

As shown in Table II, the collected data cover the mainstream tree for Linux ranging from versions 2.4 to 4.9 for all targeted products and hardware platforms. The total number of reports is 5741 and data range over a period from Nov 2002 through Nov 2016.

B. Bug Classification Approach

The bug type classification method is adopted from [8]. According to the complexity of fault triggers, a bug is classified as a Bohrbug (**BOH**) or a Mandelbug (**MAN**) [11], [12]. A Mandelbug can be further classified as a non-aging related Mandelbug (**NAM**) or an aging related bug (**ARB**). The overview of bug types is shown in Fig. 2, and the subtype definitions of NAM and ARB are listed below.

The definitions of NAM subtypes:

- **LAG**: there exists a time lag between the activation of the bug and the occurrence of its failure;
- **ENV**: the interactions of the software application with its system-internal environment have impact on the activation and/or error propagation;
- **TIM**: the timing of inputs and operations is the factor that affects the fault activation and/or error propagation;
- **SEQ**: the sequencing (i.e., the relative order) of inputs and operations is the factor that influences the activation and/or error propagation;

The definitions of ARB subtypes:

- **MEM**: the root cause of ARBs related to the accumulation of errors as a result of improper memory management (e.g., memory leaks, buffers not being flushed);
- **STO**: the root cause of ARBs related to the accumulation of errors as a result of improper storage space management (e.g., disk space is consumed by the bug);
- **LOG**: the root cause of ARBs resulting in leaks of other logical resources (system-dependent data structures e.g., inodes or sockets that are not freed after usage);
- **NUM**: the root cause of ARBs is a result of the accumulation of numerical errors (e.g., integer overflows, round-off errors);
- **TOT**: the root cause of ARBs is that the fault activation or error propagation rate increases with total system runtime, but it is not induced by accumulation of internal error states.

Fig. 2. Bug types. Based on the complexity of fault triggers, bugs are classified as Bohrbugs (BOHs) and Mandelbugs (MANs). Mandelbugs can further be categorized as non-aging related Mandelbugs (NAMs) and aging related bugs (ARBs). There are also have subtypes in NAM and ARB.

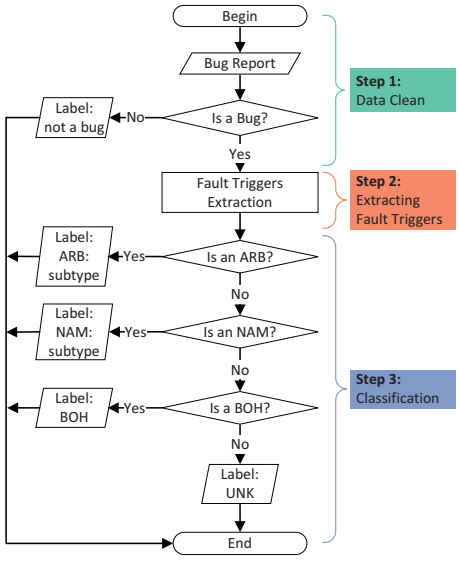

Fig. 3. Bug report classification procedure. Step 1: data clean; step 2: extracting fault triggers; step 3: classification.

C. Bug Report Classification Procedure

For a given bug report, there are three step procedure for the classification, as exhibited in Fig. 3. Each step is described in the following.

- **Step 1**: data clean. The bug report should first be examined to make sure that it was a bug, i.e., a request for new features or for enhancements, a documentation issue (e.g., missing, outdated documentation, or harmless warning output), compile-time issues (e.g., make errors or linking errors), operator errors and duplicates were removed from the analysis.
- **Step 2**: extracting fault triggers. The descriptions, discussion comments, patches, log files and other attached files of the bug report were carefully examined, to find out:

1) the activation conditions, e.g., the set of events and/or inputs needed to trigger errors; 2) the error propagation, e.g., what parameter or state of the program was changed by the bug and how a changed parameter or state propagated; 3) the manifestation of the failure, e.g., what the phenomenon would the users see when failure occurred.

- **Step 3**: classification. According to the extracted fault triggers and the failure manifestation phenomenon, as well as the characteristics of each subtype of ARB and NAM, we successively checked whether the bug belong to ARB, NAM or BOH, respectively. Note that, if a bug was classified as an ARB, but there was not enough information to extract failure mechanics, we labeled it as **ARU**. Similarly, **NAU** is labeled for the bug categorized as an NAM but lack of information to decide its activation and error propagation conditions. In the end, a report was labeled as an unknown type (**UNK**) if it did not have sufficient details to classify as ARB, NAM, or BOH.

The classification is manually implemented by the authors and when encounter suspicious classified cases, cross-checks are taken. To clarify the classification, some examples with their partial descriptions, are shown in Table III. Furthermore, our data is released to our research website[2], enabling other researchers to understand and implement the classification more easily. Among the 5741 bug reports, 4378 reports are identified as actual bugs, which account for 76.26%.

D. Correlation of Bug Types and Report Attributes

For investigating the correlation between bug types and report attributes, a statistical metric called $lift$ is implemented [30]. The $lift$ of category a_i and category b_j is calculated as $P(a_i b_j)/(P(a_i) * P(b_j))$, where $P(a_i b_j)$ is the probability that a bug belongs to both category a_i and b_j. If $lift(a_i, b_j)$ is greater than 1, categories a_i and b_j are positively correlated, which means that if a bug belongs to a_i, it is more likely to also belong to b_j. Symmetrically, if $lift(a_i, b_j)$ is less than 1, i.e., if a bug belongs to a_i, it is less likely to also belongs to b_j. In addition, if $lift(a_i, b_j)$ is equal to 1, it means the two categories a_i and b_j are not correlated.

For instance, if the total number of bugs are 100, 30 of which belong to product Drivers, 20 of which are BOHs, and 10 of which are Drivers bugs that also are BOHs, the correlation $lift$ between Drivers bugs and BOHs is calculated as follows. $P(a_i b_j)$, where a_i is Drivers bugs and b_j is BOHs, is $10/100$. $P(a_i)$ is $30/100$, and $P(b_j)$ is $20/100$. The correlation $lift(a_i, b_j) = P(a_i b_j)/(P(a_i) * P(b_j)) = (10/100)/((30/100) * (20/100)) = 1.67$, which means that a bug belonging to product Drivers is more likely a BOH.

IV. Analysis

This section reports the results of analyzing of bug reports in terms of findings and their implications. The results are the answers of the four RQs illustrated in Introduction.

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

TABLE III
Some Examples of Classified Bugs.

ID	Type	Description
6045	NAM/TIM	"Using the aic94xx/sas_class driver...,intermittent panic/hang on boot.. due to a race condition between device discovery of the root disk and an attempt to mount the root file system"
7968	BOH	"After booting (and during booting) the keyboard LEDs (NumLock, CapsLock and ScrollLock) don't work (they're always off)."
11805	NAM/ENV	"mounting XFS produces a segfault...When there is no memory left in the system, xfs_buf_get_noaddr() can fail."
12684	NAM/LAG	"After a suspend/resume, and a second suspend, the machine refuses to resume... this could be rectified by forcibly saving and restoring the ACPI non-volatile state"
50181	ARB/MEM	"After 20 hours of uptime, memory usage starts going up..."

A. Evolution of Bug Type Proportions

1) Overview of Bug Type Proportions: Fig. 4 (a) exhibits the total numbers and percentages of each bug type: BOH, NAM, ARB, as well as UNK. Note that, BOHs, NAMs and ARBs are classified bugs, which account for 92.16% of all actual bugs.

Finding #1: *Among actual bugs, 55.82% are BOHs and 36.34% are MANs.*

It can be observed from Fig. 4 (a) that, more than half of bugs in Linux are BOHs. Although BOHs are easy to reproduce and to debug once detected, there still possesses a huge proportion in Linux bugs. This might be attributed to the difficulty in testing a large operating system. Besides, the continuous development of Linux might lead to a high proportion of BOHs.

The share of MANs (NAMs and ARBs) is 36.34%, which constitutes a non-negligible part. The percentage of MANs is close to other software systems, such as 38% in MySQL [8], 36.5% in space mission on-board software [21] and 31.4% in Android [22]. Due to the complex fault triggers of MANs, specific testing method should be developed.

Implications: *For BOHs, sufficient testing should occur before releasing, such as LTP (Linux Testing Project) [31]. For MANs, as a non-negligible fraction exists, specific testing methods (e.g., combinatorial testing [32]) and cost-effective fault tolerance techniques (e.g., environment diversity [33]), should be developed to handle them.*

As shown in Fig. 4 (a), the proportions of NAMs and ARBs are 31.66%, 4.68%, respectively. In the following, we further examine the proportions of each subtype in NAMs and ARBs in detail.

Finding #2: *For NAMs, the major subtypes are TIM (37.23%), ENV (36.51%) and LAG (19.12%).*

It can be observed from Fig. 4 (b) that, the major subtypes in NAMs are TIM, ENV and LAG, which is well

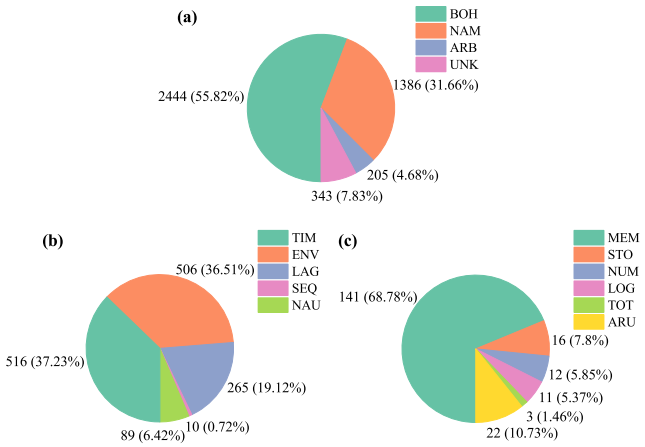

Fig. 4. Bug type proportions. (a) Total numbers and percentages for each bug type among the 4378 actual bugs. (b) Numbers and proportions of NAM subtypes. (c) Numbers and proportions of ARB subtypes.

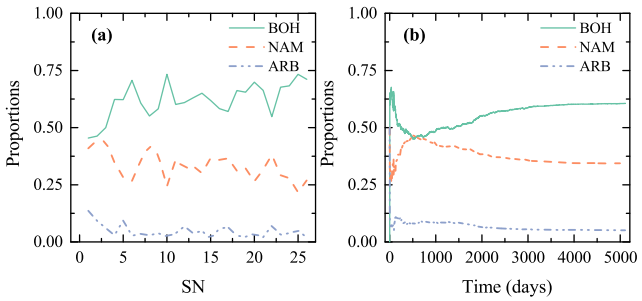

Fig. 5. Evolution of bug type proportions among classified bug reports. (a) Evolve over versions. Note, SN represents the sequential number which is assigned to each version according to their release data, e.g., the sequence number of version 2.6.15 is 1 and that for version 3.0 is 26. The symbol will be used in the remaining parts of the paper. (b) Evolve over time.

in accordance with the results of a previous study [8]. The high proportions of TIM and ENV are understandable as the Linux kernel inherently must concur concurrent activities and access shared resources (e.g., race condition: "ID-13738: Soft-Lockup/Race in networking in 2.6.31-rc1+195", deadlock: "ID-11824: raw1394: possible deadlock if accessed by multithreaded app"), as well as hardware management (e.g., "ID-8968: One broken USB storage device can hang the entire USB subsystem"). As for LAG, the root causes are often due to data corruption problems or state incorrect changing, but failures manifest only after these errors have propagated through the system. In subtype LAG, the common error is null pointer dereference (e.g., "ID-10048: ipv4/fib_hash.c: fix NULL dereference").

Implications: *To mitigate the impact of NAMs on Linux, debugging, testing or fault tolerance should focus on subtypes TIM, ENV and LAG. For testing TIM, threads conflict or locking mechanisms should be paid more attention. For testing ENV, the interaction with hardware should be focused on. For testing LAG, the values of data variables or state variables, especially those that are passed in modules or subsystems, should be examined carefully.*

Finding #3: *For ARBs, more than two thirds (68.78%) are related to MEM.*

Fig. 4 (c) displays numbers and proportions of ARB subtypes. It can be seen that, the major subtype in ARBs is MEM, which is well in accordance with the results of previous studies [8], [21], [22], [34]. As known, Linux is written in C language, in which memory management is handled by developers, which makes it more prone to software aging. Besides, leaks associated with storage, numerical problems, as well as other logical resources were also found.

Implications: *Resource release in Linux should be paid special attention by developers. For MEM, some dynamic memory bug detection tools (e.g., Kmemleak[35], a kernel memory leak detector.) and static code analysis tools (e.g., Cppcheck [36]) are suitable for memory leaks debugging.*

2) Evolution of Bug Type Proportions: According to the version integrating previously described in Section III A, the distributions of classified bugs in versions 2.4 to 4.9 are counted. To ensure the result validity, we choose versions (i.e., versions 2.6.15 to 3.0) that have more than 50 bugs to analyze the evolution of bug type proportions over versions, as shown in Fig. 5 (a). Moreover, we analyze the evolution of bug type proportions with reported time. Due to the overlapping life cycle of two major version series (e.g., version 3.7 series was maintained from Dec 2012 to Mar 2013, while version 3.8 series was maintained from Feb 2013 to May 2013.), we consider all versions in the temporal analysis in the following, as exhibited in Fig. 5 (b).

Finding #4: *The proportion of BOHs tends to slowly increase with the evolution of versions or time, while the proportion of NAMs tends to slowly decrease. For ARBs, its proportion tends to slightly decrease over time. For all three types, the proportions stabilize around a constant value after 4000 days.*

The evolution trends of the proportions for BOHs, NAMs and ARBs in Fig. 5 (a), are tested by means of Mann-Kendall trend test [37], [38] and the results indicate that the evolution trends of proportions for BOHs and NAMs are significant at the 5% ($\alpha = 0.05$) level, but for ARBs is insignificant.

About every two months, a major version of Linux would be released (e.g., version 4.1 was released on Jun 22, 2015, while version 4.2 was put out on Aug 30, 2015). With the evolution of Linux, the complexity keeps continuously growing (i.e., the number of functions [14], and the lines of code [15], continuously grow with the development of Linux), which is results of a massive number of features being introduced. This might lead to more BOHs in newly released versions. Although code changes would also introduce NAMs, the slow decreasing proportion of NAMs might be attributed to the fast growth rate of BOHs proportion. Moreover, the proportion of ARBs tends to be more stable with the evolution of versions or time.

Implications: *For frequent-release characteristic of Linux*

development paradigm, developers are suggested to pay greater attention to bugs introduced in new features, and a continuous functional testing activity is required.

3) Comparison of Evolutions of Bug Type Proportions Across Four Versions: In this part, four versions including versions 2.6.0, 2.6.24, 2.6.27 and 2.6.32, that possess the most bugs in all versions, are chosen to examine the evolution of bug type proportions over the life time of a version, as displayed in Fig. 6.

Finding #5: *The proportions of bug types and their evolution trends are different across versions. For all selected versions, the proportions of BOHs and MANs tend to stabilize around a constant value after 600 days.*

As depicted in Fig. 6 (a), the proportion of MANs is greater than BOHs in version 2.6.0, the first version of 2.6 series. This might be due to the introducing of a new CPU scheduler. In previous versions, all runnable tasks were kept on a single run-queue that represented a linked list of threads. However, in version 2.6.0, the single run-queue lock was replaced with a per CPU lock, which means that the kernel is preemptive: it can immediate response to interactive processes [39]. This feature might cause more NAMs, especially more timing-related bugs (e.g., race conditions and deadlocks), since developers need time to adapt to the new scheduler.

Moreover, it can be observed from Fig. 6 (c) that in version 2.6.27, the proportion of MANs tends to increase after 750 days. This might be attributed to the reason that version 2.6.27 is one of long-term support versions. They are a type of special versions which are supported for a very long period. Long-term support version could become more stable during the maintenance, thus the proportion of easily isolated and reproduced bugs (i.e., BOHs) would decrease, while difficult to fix bugs (i.e., MANs) would increase.

Implications: *The evolution trends of bug type proportions can be utilized as indicators to decide which testing strategies should be implemented for each version.*

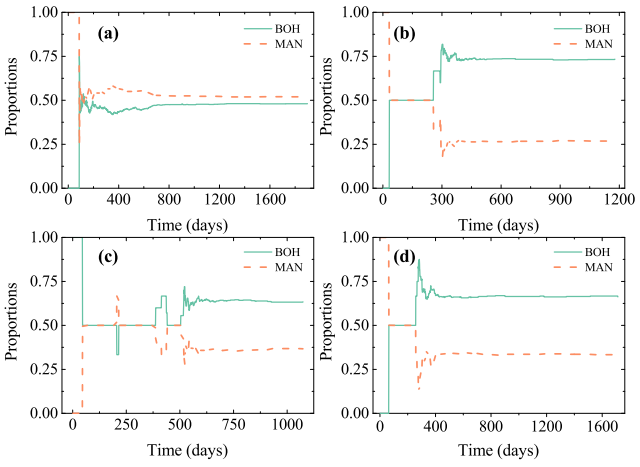

Fig. 6. Evolution of bug type proportions of four selected versions, including (a) 2.6.0, (b) 2.6.24, (c) 2.6.27 and (d) 2.6.32.

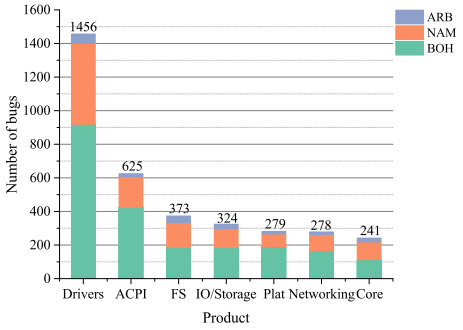

Fig. 7. Distributions of classified bugs in products. FS denotes File system, Plat stands for Platform Specific/Hardware, Core includes bugs in 3 products: Memory Management, Process Management and Timers.

4) Evolution of Bug Type Proportions in Products: To understand the impact of different products on bug types, we analyze proportion distributions of bug types and temporal evolution in products. In Linux kernel Bugzilla, bugs are categorized into products (e.g., Drivers, File System, Memory Management, Process Management, etc.), which refer to different specific functions of Linux. Fig. 7 depicts the distributions of classified bugs in the selected products that possessing the most number of bugs. Note, the numbers of BOHs, NAMs and ARBs in these products account for 89.32%, 87.37% and 88.78%, of the total numbers of BOHs, NAMs and ARBs, respectively.

Finding #6: *The number of bugs in products related to Drivers (Drivers and ACPI) accounts for 51.57% of all classified bugs.*

Linux supports a large number of devices, e.g., more than 100 types of devices in version 4.1 are supported and the number of functions in their source codes of drivers account for around 50% of the total functions of Linux [14]. In addition, product ACPI is short for advanced configuration and power interface, which is closely related to hardware devices. Due to a great diversity of device support, it is difficult to perform compatibility testing for all device drivers.

Implications: *In Linux testing, more attention should be paid to Drivers, since more than half of the classified bugs are related to device drivers.*

To further analyze the correlation between bug types and products, we use a metric named $lift$ (as described in Section III D) to investigate which bug types would more likely occur in each product. Table IV shows the correlation between bug types and products.

Finding #7: *In products, a bug belonging to Drivers, ACPI or Platform is more likely a BOH; a bug occurring in File System, IO/Storage or Core (Memory, Process Management and Timers) is more likely an NAM or ARB; a Networking bug is more possible an NAM.*

The different bug type manifestations in products might be due to different characteristics of products. As for products Drivers, ACPI and Platform Specific/Hardware, although these products are regarded as bridges between an operating system

TABLE IV
CORRELATION BETWEEN BUG TYPES AND PRODUCTS.

	BOH	NAM	ARB
Drivers	**1.03**	0.98	0.74
ACPI	**1.11**	0.87	0.53
FS	0.82	**1.20**	**1.90**
IO/Storage	0.94	**1.69**	**1.52**
Plat	**1.12**	0.79	0.92
Networking	0.98	**1.04**	0.99
Core	0.77	**1.30**	**1.79**

and devices, failures in these products manifest more directly to the user site. When users using a device, if its drivers were not coded correctly, the device would not be functional to users. Therefore, bugs occurred in these products would be more likely BOHs.

While for products File System, IO/Storage, Networking and Core (Memory, Process Management and Timers), these products are regarded as basic or core functions from an operating system aspect. This means that the interactions among these products are more complex and tight [40]. Consequently, bugs related to these products are more prone to be MANs.

Implications: *Different testing strategies should be implemented to different products. For instance, for testing product Drivers, more functional testing and compatibility testing should be taken. While for testing product File System, Networking, IO/Storage, etc., combinatorial testing might be useful [32], since bugs related to these products are more likely MANs.*

In addition, we explore the evolutions of bug type proportions of the selected products in the following, as depicted in Fig. 8.

Finding #8: *Evolution trends of bug type proportions are different in products. The proportions of NAMs in products File System, IO/Storage and Core (Memory, Process Management and Timers) tend to slightly increase with time. The proportions of BOHs tend to slowly increase with time in all products. For ARBs, the proportions tend to stabilize around a constant value after about 3000 days.*

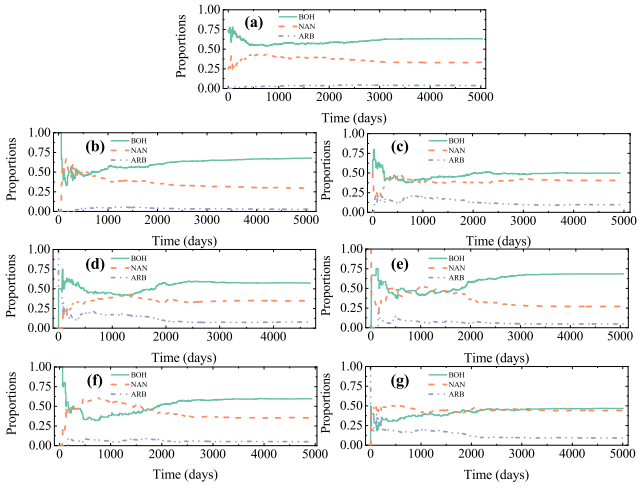

Fig. 8. Evolution of bug type proportions of selected products, including (a) Drivers, (b) ACPI, (c) FS, (d) IO/Storage, (e) Plat, (f) Networking and (g) Core (Memory, Process Management and Timers).

TABLE V
TWO EXAMPLES OF REGRESSION BUGS AND THEIR BUG TYPES.

ID	Type	Description
8736	NAM/TIM	"Here is another scenario I bumped onto - qdisc_watchdog_cancel() and qdisc_restart() deadlock ...Please try reverting commit 1936502d0. This one is a regression in 2.6.22"
11329	BOH	"in git1 and previous, cpu0_vid is reported as 1475 (which is correct). Since git2, it is reported, as 725"

Implications: *refer to implications for Findings #4 and #7.*

B. Evolution of Regression bugs

A regression bug in Linux means that a bug causes an existing feature, which used to work, to fail or misbehave completely in recent versions [16]. Two examples of regression bugs and their bug types are exhibited in Table V.

1) Proportions of Bug Types in Regression Bugs: In this part, we make a statistic of numbers of regression bugs and non-regression bugs, as exhibited in Fig. 9 (a).

Finding #9: *More than half of the classified bugs are regression bugs.*

It can be observed from Fig. 9 (a) that, more than half of the classified bugs in Linux are the existing feature broken problems, which means regression bugs. The proportion of regression bugs (50.06%) in Linux is similar with other software systems, such as 51.09% in Google Chromium [29]. Furthermore, we perform a comparison of bug type proportions between regression and non-regression bugs, as displayed in Fig. 9 (b).

Finding #10: *The proportion of BOHs in regression bugs is higher than that in non-regression bugs. Moreover, a regression bug is more likely a BOH, while a non-regression bug is more possible an NAM or ARB.*

We can observe from Fig. 9 (b), there are more BOHs in regression bugs, while there are more MANs in non-regression bugs. To further analyze the correlation between bug types and regressions, we use a metric named $lift$ (as described in Section III D) to investigate which bug types would more likely occur in regression or non-regression categories, as shown in Table VI. A regression bug is more likely a BOH, but a non-regression bug is more possible a MAN. This indicates that code changes (e.g., new feature introducing and bug fixing) would bring more BOHs rather than MANs. Regressions are annoying since they make it harder for users to upgrade the kernel. As a result, users with old kernels would more likely suffer security problems.

Implications: *Developers are suggested to implement more regression testing before releasing a new version, to reduce*

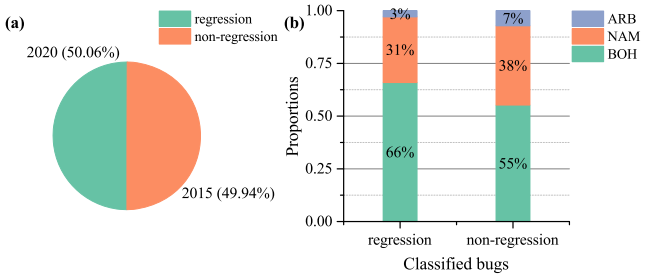

Fig. 9. Regression and non-regression bugs among classified bug reports. (a) Numbers and percentages for regression and non-regression bugs. (b) Comparison of bug types for regression and non-regression bugs.

TABLE VI
CORRELATION BETWEEN BUG TYPES AND REGRESSIONS.

	BOH	NAM	ARB
regression	**1.09**	0.91	0.58
non-regression	0.91	**1.09**	**2.00**

the existing feature broken problems, since a half of bugs are regression bugs. While dealing with non-regression bugs, specific testing methods (e.g., combinatorial testing [32]) would be more effective, since a non-regression bug is more likely a MAN.

2) Evolution of the Proportion of Regression bugs: In the following, the evolution trends of proportions of regression bugs and non-regressions are explored. The evolution analysis over version and time are exhibited in Fig. 10 (a) and (b), respectively.

Finding #11: *The proportion of regression bugs tends to grow with the evolution of versions or time, and it stabilizes around a constant value 0.5 after 3500 days.*

During the evolution of Linux, several new features are introduced. According to a previous study of Linux evolution [14], the numbers of supported types of device drivers, file systems, networking protocols, architecture platforms in version 2.4 are less than 40, 40, 30 and 15, respectively, but increase to more than 110, 60, 50 and 25, respectively in version 4.1. Significant number of features integrating into Linux would bring a massive number of code changes. Therefore, it would be inevitable to cause existing feature broken problems. In addition, bug fixing is also the reason (e.g., "ID-10232: intel mtrr fixups apparently broke display and e1000 probe" and "ID-43228: Commit 1cc0c998 (ACPI: Fix D3hot v D3cold confusion) breaks turning off nvidia optimus card", etc.). Thus, the results of Finding #11 are expected and give evidences of Findings #1 and #4.

Implications: *Since regression could bring more BOHs in Linux, it should be more careful to take code changes (e.g., feature adding or bug fixing), and continuous regression testing should be done before releasing a new feature or fixing a bug [16].*

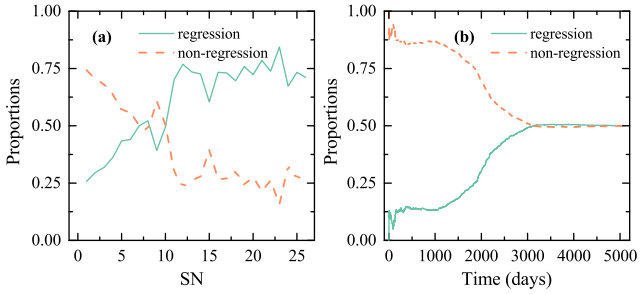

Fig. 10. Evolution of proportions of regression and non-regression bugs among classified bug reports. (a) Evolve over versions. (b) Evolve over time.

C. Evolution of Bug Type Proportions with Software Metrics

Software systems represent one of the most sophisticated man-made systems which can be modeled as networks [41], [42], [43], [44]. In this section, we analyze the relationship between bug type proportions and a complex network metric clustering coefficient. Clustering coefficient is used to measure the tendency of a network to form tightly connected neighborhoods, and its definition can be seen in [14]. To ensure the result validity, in this part we choose those versions with more than 50 bugs for analyzing. The relationships between bug type proportions and clustering coefficient are depicted in Fig. 11 (a) and (b), respectively. Besides, Pearson correlation analysis is shown in Table VII.

Finding #12: *With the evolution of clustering coefficient, Linux with a large clustering coefficient tends to have a low proportion of BOHs. In contrast, Linux with a large clustering coefficient tends to possess a high proportion of MANs.*

As depicted in Table VII, correlations are significant at the 0.01 level (note, the sample size n is 32). A strong negative correlation between the proportion of BOHs and clustering coefficient is observed. Note that, according to the result of the evolution of Linux network, clustering coefficient decreases with the evolution of versions [14]. This means that with the evolution of clustering coefficient, the proportion of BOHs tends to become higher. By contrast, the proportion of MANs tends to be smaller, due to a strong positive correlation with clustering coefficient, as exhibited in Table VII. This might be due to the reason that, a large clustering coefficient means

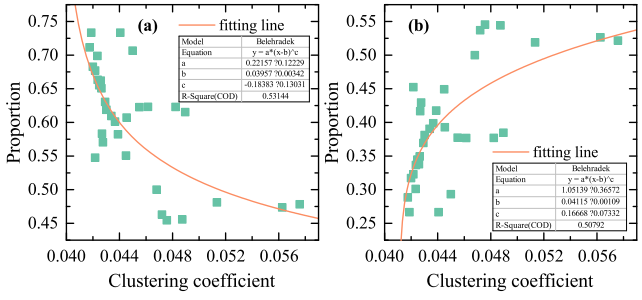

Fig. 11. The relationship between bug type proportions and clustering coefficient. (a) BOH, (b) MAN.

TABLE VII
PEARSON CORRELATION ANALYSIS BETWEEN BUG TYPE PROPORTIONS AND CLUSTERING COEFFICIENT.

	Clustering coefficient	p value
BOH	-0.68	0.000022
MAN	0.68	0.000022

there exists a tight local connection of functions, which could bring more interactions among internal functions. This could lead to a more complex failure manifestation (i.e., MANs). In contrast, a small clustering coefficient refers to a loose local connection of functions in terms of a less interaction among them, which might bring a less complex failure manifestation (i.e., BOHs).

Implications: *Clustering coefficient can be utilized as an indicator to measure the bug type proportions in Linux.*

D. The Relationship Between Bug Types and Time to Fix

In this section, we investigate the relationship between bug types and time to fix the bug.

According to the definitions of BOH and MAN, the fault triggers of a MAN are more complex than that of a BOH. Therefore, it is supposed to have longer time to fix a MAN. Since there is no fixing time recorded in bug reports, we estimate the fixing time by the difference between the reported time and the last modified time.

Finding #13: *The average time taken to fix a MAN tends to be longer than that for fixing a BOH.*

The average fixing time and standard deviation of two bug types are shown in Table VIII. It can be notably observed that the average time taken to fix a BOH is 218.63 days, while for a MAN is 254.22 days. The result is further to be verified by means of the Wilcoxon-Mann-Whitney test [45]. The null hypothesis is that for both types of bugs the fixing time is sampled from the same distribution. For a given criteria ($\alpha = 0.05$), after performing the test, we got that the p value is 0.00002, which means that the null hypothesis can be rejected at 95% of confidence. Therefore, we can draw a conclusion that it is more likely to take a longer time to fix a MAN than fix a BOH. This shows a well in accordance with previous studies in HTTPD [8], AXIS [8], and Android [22].

A bug report from its submission to the final resolution would go through several states, including Unconfirmed, New, Assigned, Resolved, Verified and Closed [1]. Once the bug is assigned to a kernel developer, the state of the bug transits to the Assigned state. The major difference of fixing time between BOHs and MANs might be attributed to the difference of transition time from states Assigned to Resolved. Due to the complexity difference between MANs and BOHs, to resolve a MAN, developers usually require sufficient information to detect the underlying root causes in the code. Besides, the non-deterministic characteristic of MANs could lead to more time taken to reproduce. Consequently, longer fixing time is needed to fix a MAN.

TABLE VIII
COMPARISON OF TIME TO FIX FOR BUG TYPES.

Bug type	Average time to fix (days)	Standard deviation
BOH	218.63	360.72
MAN	254.22	374.22

Implications: *For MANs, specific testing methods and fault tolerance approaches should be implemented, due to their long fixing time.*

V. THREATS TO VALIDITY

As other empirical studies, our study is naturally subject to limitations. We identify the following threats:

Selection of bug reports: we only analyze fixed and closed reports, since unfixed and unclosed reports may contain incomplete information. Bug type proportions could be different if unfixed and unclosed reports are considered.

Manual inspection: although we have examined carefully all the related information in bug reports, involving descriptions, comments, as well as attached files and patches, classification mistakes could not be avoided.

Evolution analysis: for version evolution analysis, although minor releases were integrated into major releases according to Linux version numbering method, the results may be influenced by the accuracy and completeness of reports. While for temporal evolution analysis, bug reporting of a release might decrease because of a new major release or fixes taking longer.

Regression bugs: the determination of regression bugs is not only according to their tags but also examining descriptions and comments, but there might still exist some mistakes due to the accuracy and completeness of reports.

Fixing time: the time to fix a bug is computed as the time when the report opens until it is closed, but it might exist some situations that reporters misused the tracking tool.

VI. CONCLUSION

In this paper, we performed an empirical investigation of bugs on Linux operating system in terms of fault triggers. With the bug classification results based on 5741 real bug reports, our analyses were conducted from an evolution perspective on four research questions: bug types in Linux, regression bugs, software metrics and time to fix, along with several interesting findings and implications that can be adopted by developers and users. Future research on Linux can benefit from our study. For instance, as to the products of Linux, File System, IO/Storage, Networking and Core (Memory Management, Process Management and Timers) should be firstly examined to reduce non-deterministic behaviors. Besides, the clustering coefficient could be utilized as an indicator to measure bug type proportions in Linux.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61772055, 61402027 and

61572150) and in part by US NSF (Grant No. CNS-1523994). The authors would like to thank Dr. Wen-Bo Du from Beihang University for kindly providing valuable technical advices to this manuscript.

REFERENCES

[1] M. F. Ahmed and S. S. Gokhale, "Linux bugs: Life cycle, resolution and architectural analysis," *Information and Software Technology*, vol. 51, no. 11, pp. 1618–1627, 2009.

[2] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2fix: Automatically generating bug fixes from bug reports," in *Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 282–291.

[3] Y. Jiang, B. Adams, and D. M. German, "Will my patch make it? and how fast? case study on the linux kernel," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 101–110.

[4] I. Abal, C. Brabrand, and A. Wasowski, "42 variability bugs in the linux kernel: a qualitative analysis," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. ACM, 2014, pp. 421–432.

[5] Y. Zhao, F. Zhang, E. Shihab, Y. Zou, and A. E. Hassan, "How are discussions associated with bug reworking?: An empirical study on open source projects," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 21.

[6] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 523–526.

[7] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1665–1705, 2014.

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

[9] D. Wijayasekara, M. Manic, and M. McQueen, "Vulnerability identification and classification via text mining bug databases," in *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE*. IEEE, 2014, pp. 3612–3618.

[10] 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.

[11] 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.

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

[13] A. Pfening, S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi, "Optimal software rejuvenation for tolerating soft failures," *Performance Evaluation*, vol. 27, pp. 491–506, 1996.

[14] G. Xiao, Z. Zheng, and H. Wang, "Evolution of linux operating system network," *Physica A: Statistical Mechanics and its Applications*, vol. 466, pp. 249–258, 2017.

[15] A. Israeli and D. G. Feitelson, "The linux kernel as a case study in software evolution," *Journal of Systems and Software*, vol. 83, no. 3, pp. 485–501, 2010.

[16] J. Johnson, J. Kenefick, and P. Larson, "Hunting regressions in gcc and the linux kernel," 2004.

[17] "I. s. 1044-2009, standard classification for software anomalies," 2010.

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

[19] 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.

[20] M. Grottke, R. Matias, and K. S. Trivedi, "The fundamentals of software aging," in *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*. IEEE, 2008, pp. 1–6.

[21] M. Grottke, A. P. Nikora, and K. S. Trivedi, "An empirical investigation of fault types in space mission system software," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. IEEE, 2010, pp. 447–456.

[22] F. Qin, Z. Zheng, X. Li, Y. Qiao, and K. S. Trivedi, "An empirical investigation of fault triggers in android operating system," in *Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on*. IEEE, 2017, pp. 135–144.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] D. Nir, S. Tyszberowicz, and A. Yehudai, "Locating regression bugs," in *Haifa Verification Conference*. Springer, 2007, pp. 218–234.

[28] E. Shihab, A. E. Hassan, B. Adams, and Z. M. Jiang, "An industrial study on the risk of software changes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 62.

[29] M. Khattar, Y. Lamba, and A. Sureka, "Sarathi: Characterization study on regression bugs and identification of regression bug inducing changes: A case-study on google chromium project," in *Proceedings of the 8th India Software Engineering Conference*. ACM, 2015, pp. 50–59.

[30] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[31] P. Larson, "Testing linux® with the linux test project," in *Ottawa Linux Symposium*, 2002, p. 265.

[32] Z. B. Ratliff, D. R. Kuhn, R. N. Kacker, Y. Lei, and K. S. Trivedi, "The relationship between software bug type and number of factors involved in failures," in *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 119–124.

[33] 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.

[34] 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.

[35] J. Corbet, "Detecting kernel memory leaks," *https://lwn.net/Articles/187979/*, 2006.

[36] D. Marjamäki, "Cppcheck: a tool for static c/c++ code analysis," *available at: http://cppcheck.sourceforge.net (last access: 30 May 2014)*, 2013.

[37] H. B. Mann, "Nonparametric tests against trend," *Econometrica: Journal of the Econometric Society*, pp. 245–259, 1945.

[38] M. G. Kendall, "Rank correlation methods." 1948.

[39] L. A. Torrey, J. Coleman, and B. P. Miller, "A comparison of interactivity in the linux 2.6 scheduler and an mlfq scheduler," *Software: Practice and Experience*, vol. 37, no. 4, pp. 347–364, 2007.

[40] H. Wang, Z. Chen, G. Xiao, and Z. Zheng, "Network of networks in linux operating system," *Physica A: Statistical Mechanics and its Applications*, vol. 447, pp. 520–526, 2016.

[41] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Physical Review E*, vol. 68, no. 4, p. 046116, 2003.

[42] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 687–708, 2007.

[43] Y. Gao, Z. Zheng, and F. Qin, "Analysis of linux kernel as a complex network," *Chaos, Solitons & Fractals*, vol. 69, pp. 246–252, 2014.

[44] H. Wang and G. Xiao, "Analysis of the runtime linux operating system as a complex weighted network," in *Software Analysis, Testing and Evolution (SATE), International Conference on*. IEEE, 2016, pp. 7–11.

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