Works For Me! Characterizing Non-reproducible Bug Reports

Mona Erfani Joorabchi Mehdi Mirzaaghaei Ali Mesbah

Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
{merfani, mehdi, amesbah}@ece.ubc.ca

ABSTRACT

Bug repository systems have become an integral component of software development activities. Ideally, each bug report should help developers to find and fix a software fault. However, there is a subset of reported bugs that is not (easily) reproducible, on which developers spend considerable amounts of time and effort. We present an empirical analysis of non-reproducible bug reports to characterize their rate, nature, and root causes. We mine one industrial and five open-source bug repositories, resulting in 32K non-reproducible bug reports. We (1) compare properties of non-reproducible reports with their counterparts such as active time and number of authors, (2) investigate their life-cycle patterns, and (3) examine 120 *Fixed* non-reproducible reports. In addition, we qualitatively classify a set of randomly selected non-reproducible bug reports (1,643) into six common categories. Our results show that, on average, non-reproducible bug reports pertain to 17% of all bug reports, remain active three months longer than their counterparts, can be mainly (45%) classified as "Interbug Dependencies", and 66% of *Fixed* non-reproducible reports were indeed reproduced and fixed.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Measurement

Keywords

Non-reproducible bugs, mining bug reports, bug tracking systems

1. INTRODUCTION

When a failure is detected in a software system, a bug report is typically filed through a bug tracking system. The developers then try to validate, locate, and repair the reported bug as quickly as possible. In order to validate the existence of the bug, the first step developers take is often using the information in the bug report to *reproduce* the failure. However, reproducing reported bugs is not always straightforward. In fact, some reported bugs are difficult or impossible to reproduce. When all attempts at reproducing a reported bug are futile, the bug is marked as *non-reproducible (NR)* [1, 5].

Non-reproducible bugs are usually frustrating for developers to deal with [9]. First, developers usually spend a considerable amount of time trying to reproduce them, without any success. Second, due to the very nature of these bug reports, there is typically no coherent set of policies to follow when developers encounter such bug reports. Third, because they cannot be reproduced, developers are reluctant to take responsibility and close them.

Mistakenly marking an important bug as non-reproducible and ignoring it, can have serious consequences. An example is the recent security vulnerability found in Facebook [17], which allowed anyone to post to other users' walls. Before exposing the vulnerability, the person who had detected the vulnerability had filed a bug report. However, the bug was ignored by Facebook engineers: ❝*Unfortunately your report [...] did not have enough technical information for us to take action on it. We cannot respond to reports which do not contain enough detail to allow us to reproduce an issue.*❞

Researchers have analyzed bug repositories from various perspectives including bug report quality [15], prediction [20], reassignment [21], bug fixing and code reviewing [13, 30], reopening [31], and misclassification [23]. None of these studies, however, has analyzed non-reproducible bugs in isolation. In fact, most studies have ignored non-reproducible bugs by focusing merely on the *Fixed* resolution.

In this paper, we provide an empirical study on non-reproducible bug reports, characterizing their prevalence, nature, and root causes. We mine six bug repositories and employ a mixed-methods approach using both quantitative and qualitative analysis. To the best of our knowledge, we are the first to study and characterize non-reproducible bug reports.

Overall, our work makes the following main contributions:

- We mine the bug repositories of one proprietary and five open source applications, comprising 188,319 bug reports in total; we extract 32,124 non-reproducible bugs and quantitatively compare them with other resolution types, using a set of metrics;

MSR '14, May 31 – June 1, 2014, Hyderabad, India

- We qualitatively analyze root causes of 1,643 non-reproducible bug reports to infer common categories of the reasons these reports cannot be reproduced. We systematically classify 1,643 non-reproducible bug reports into the inferred categories;
- We extract patterns of status and resolution changes pertaining to all the mined non-reproducible bug reports. Further, we manually investigate 120 of these non-reproducible reports that were marked as *Fixed* later in their life-cycle.

Our results show that, on average:

1. NR bug reports pertain to 17% of all bug reports;
2. compared with bug reports with other resolutions, NR bug reports remain active around *three months* longer, and are similar in terms of the extent to which they are discussed and/or the number of involved parties;
3. NR bug reports can be classified into 6 main cause categories, namely "Interbug Dependencies"' (45%), "Environmental Differences" (24%), "Insufficient Information" (14%), "Conflicting Expectations" (12%), and "Non-deterministic Behaviour" (3%);
4. 68% of all NR bug reports are resolved directly from the initial status (New/Open). The remaining 32% exhibit many resolution transition scenarios.
5. NR bug reports are seldom marked as *Fixed* (3%) later on; from those that are finally fixed, 66% are actually reproduced and fixed through code patches (i.e., changes in the source code).

2. NON-REPRODUCIBLE BUGS

Most bug tracking systems are equipped with a default list of bug *statuses* and *resolutions*, which can be customized if needed. Generally, each bug report has a *status*, which specifies its current position in the bug report life cycle [5]. For instance, reports start at *New* and progress to *Resolved*. From *Resolved*, they are either *Reopened* or *Closed*, i.e., the issue is complete. At the *Resolved* status, there are different *resolutions* that a bug report can obtain, such as *Fixed*, *Duplicate*, *Won't Fix*, *Invalid*, or *Non-Reproducible* [5, 1].

There are various definitions available for non-reproducible bugs online. We adopt and slightly adapt the definition used in Bugzilla [1]:

Definition 1 *A* Non-Reproducible (NR) bug *is one that cannot be reproduced based on the information provided in the bug report. All attempts at reproducing the issue have been futile, and reading the system's code provides no clues as to why the described behaviour would occur.*

Other resolution terminologies commonly used for non-reproducible bugs include *Cannot Reproduce* [11], *Works on My Machine* [12] and *Works For Me* [10].

Our interest in studying NR bugs was triggered by realizing that developers spend considerable amounts of time and effort on these reports. For instance, issue #106396 in the Eclipse project has 62 comments from 28 people, discussing how to reproduce the reported bug [2]. This motivated us to conduct a systematic characterization study of non-reproducible bug reports to better understand their nature, frequency, and causes.

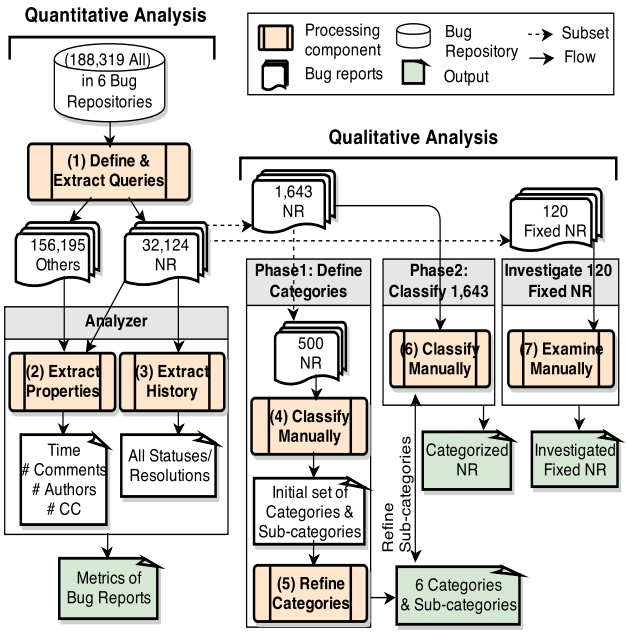

Figure 1: Overview of our methodology.

3. METHODOLOGY

Our analysis is based on a mixed-methods research approach [18], where we collect and analyze both quantitative as well as qualitative data. All our empirical data is available for download [8]. We address the following research questions in our study:

RQ1. How prevalent are NR bug reports? Are NR bug reports treated differently than other bug reports?

RQ2. Why can NR bug reports not be reproduced? What are the most common cause categories?

RQ3. Which resolution transition patterns are common in NR bug reports?

RQ4. What portion of NR bug reports is fixed eventually? Were they mislabelled initially? What cause categories do they belong to?

Figure 1 depicts our overall approach. We use this figure to illustrate our methodology throughout this section.

3.1 Bug Repository Selection

To answer our research questions, we need bug tracking systems that provide advanced search/filter mechanisms and access to historical bug report life-cycles. Since Bugzilla and Jira both support these features (e.g., `Changed to/from` operators), we choose projects that use these two systems.

Table 1 shows the bug repositories we have selected for this study. To ensure representativeness, we select five popular, actively maintained software projects from three separate domains, namely *desktop* (Firefox and Eclipse), *web* (MediaWiki and Moodle), and *mobile* (Firefox Android). In addition, we include one commercial closed source application (Industrial). The proprietary bug tracking system is from a Vancouver-based mobile app development company. The bug reports are filed by their testing team and end-users,

Table 1: Studied bug repositories and their rate of NR bugs.

ID	Domain	Repository	Product/Component	#All Bugs*	#NR Bugs**	NR(%)	FixedNR(%)***
FF	Desktop	Bugzilla [3]	Firefox	65,408	18,516	**28%**	1%
E	Desktop	Bugzilla [4]	Eclipse/Platform	65,475	8,189	**13%**	4%
W	Web	Bugzilla [6]	MediaWiki	9,335	1,125	**12%**	9%
M	Web	Jira [7]	Moodle	22,175	2,503	**11%**	5%
FFA	Mobile	Bugzilla [3]	FirefoxAndroid	7,902	1,148	**15%**	3%
PTY	Mobile	Jira	Proprietary	18,024	643	**4%**	17%
Overall				188,319	32,124	**17%**	3%

*All_Query: Resolution: All except (*Duplicate*, *Invalid*, *Rejected*) and Severity: All except (*Enhancement*, *Feedback*) and Status: All except *Unconfirmed*
**NR_Query: All_Query and Resolution: `Changed to/from` *Non-Reproducible*
***FixedNR_Query: Resolution: *Fixed* and Severity: All except (*Enhancement*, *Feedback*) and Status: All except *Unconfirmed* and Resolution CHANGED FROM *Non-Reproducible* and Resolution: CHANGED TO *Fixed*

and are related to different mobile platforms such as Android, Blackberry, iOS, and Windows Phone, as well as their content management platform and backend software.

3.2 Mining Non-Reproducible Bug Reports

In this study, we include all bug reports that are resolved as non-reproducible at least once in their life-cycles. In our search queries, we include all resolution terminologies commonly used for non-reproducible bug reports, as outlined in Section 2. We extract these NR bug reports in three main steps (Box 1 in Figure 1):

Step 1. We start by filtering out all *Invalid*, *Duplicate*, and *Rejected* reports. Where applicable, we also exclude *Enhancement*, *Feedback*, and *Unconfirmed* reports. The set of bug reports retrieved afterward is the total set that we consider in this study ('#All Bugs' in Table 1).

Step 2. We use the filter/search features available in the bug repository systems and apply the `Changed to/from` operator on the resolution field to narrow down the list of bug reports further to the non-reproducible resolution ('#NR Bugs' in Table 1).

Step 3. We extract and save the data in XML format, containing detailed information for each retrieved bug report.

This mining step was conducted during August, 2013. We did not constrain the start date for any of the repositories. The detailed search queries used in our study are available online [8]. Overall, our queries extracted **32,124** NR bug reports from a total of 188,319 bug reports.

3.3 Quantitative Analysis

In order to perform our quantitative analysis, we measure the following metrics from each extracted bug report:

Active Time pertains to the period between a bug report's creation and the last update in the report.

Number of Unique Authors measures the number of people directly involved with the report, based on their user ID.

Number of Comments provides information about the extent to which a bug is discussed; this is an indication of how much attention a bug report attracts.

Number of CCs/Watchers measures the number of people that would receive update notifications for the report. It provides insights as how many people are interested in a particular bug report.

Historical Status and Resolution Changes collects data on how the status and resolution of a bug report changes throughout time.

To address RQ1, we measure the first four metrics for all the bug reports to compare the properties of *NR* bug reports (32,124) with the others (156,195). We built an analyzer tool, called NR-Bug-Analyzer [8], to calculate these metrics. It takes as input the extracted XML files and measures the first four metrics (Box 2 in Figure 1). Since each repository system has a different set of fields, we performed a mapping to link common fields in BUGZILLA and JIRA, as presented in Table 2.

To address RQ3, the last metric (historical changes) is extracted for all NR bug reports and used to mine common transition patterns. The data retrieved from bug repositories does not contain any information on how the statuses and resolutions change over time for each bug report. Thus our tool parses the HTML source of each NR bug report to extract historical data of status and resolution changes (Box 3 in Figure 1). BUGZILLA provides a *History Table* with historical changes to different fields of an issue, including the status and resolution fields, attachments, and comments. We extract the history of each bug report by concatenating the issue ID with the base URL of the HISTORY TABLE.[1] JIRA provides a similar mechanism called *Change History*. Our bug report analyzer tool along with all the collected (open source) empirical data are available for download [8].

3.4 Qualitative Analysis

In order to address RQ2, we perform a qualitative analysis that requires manual inspection. To conduct this analysis in a timely manner, we constrain the number of NR bug reports to be analyzed through random sampling. The manual classification is conducted in two steps, namely, common category inference and classification.

Common Category Inference. In the first phase, we aim to infer a set of common categories for the causes of NR bugs, i.e., understanding why they are resolved as NR. We randomly selected 250 NR reports from the open source repositories and 250 NR reports from INDUSTRIAL.

In order to infer common cause categories, each bug report was thoroughly analyzed based on the bug's description, tester/developer discussions/comments, and historical data. We defined a set of classification definitions and rules and generated the initial set of categories and sub-categories (Box

[1]For example, the base URL for the *History Table* in FIREFOX BUGZILLA is `https://bugzilla.mozilla.org/show_activity.cgi?id=bug_id`.

Table 2: Mapping of BUGZILLA and JIRA fields.

#	Bugzilla	Jira	Description
1	bug_id	key	The bug ID.
2	comment_id	id (in *comment* field)	A unique ID for a comment.
3	who	author (in *comment* field)	Name and id of the user who added a bug, a comment, or any other type of text.
4	creation_ts	created	The date/time of bug creation.
5	delta_ts	resolved (updated)	The timestamp of the last update. If *resolved* field is not available, *updated* field is used.
6	bug_status	status	The bug's latest status.
7	resolution	resolution	The bug's latest resolution.
8	cc	watches	Receive notifications.

4 in Figure 1). Then, the generated (sub)categories were cross validated through discussions, merged, and refined (Box 5 in Figure 1). Based on an analysis of the reasons the bug reports could not be reproduced, in total, we extracted six high level cause categories, each with a set of sub-categories, which were fed into our classification step. The categories and our classification rules are presented in Table 3. In the given examples in Table 3 and throughout the paper, R refers to reporter and D refers to anyone else other than reporter.

Classification. In the second phase, we randomly selected 200 NR bug reports from each of the open source repositories. In addition, to have a comparable number of NR bug reports from the commercial application, we included all the 643 NR bug reports from INDUSTRIAL in this step. We then systematically classified these 1,643 NR bug reports, using the rules and (sub)categories inferred in the previous phase. Where needed, the sub-categories were refined in the process (Box 6 in Figure 1). Similar to the category inference step, each bug report was manually classified by analyzing its descriptions, discussions/comments, and historical activities. At the end of this step, each of the 1,643 NR bug reports was distributed into one of the 6 categories of Table 3.

Inspecting *Fixed* NR Bug Reports. To address RQ4, we performed a query on the set of NR bug reports to extract the subset that is finally changed to a Fixed resolution.

We randomly selected 20 fixed NR bug reports from the 6 repositories and manually inspected them (120) to understand why they were marked as *Fixed* (Box 7 in Figure 1), to understand whether the reports were initially mislabelled [23] or became reproducible/fixable, e.g., through additional information provided by the reporter. In addition, this would provide more insights in types of NR bug reports that are expected to be fixed, and the additional information that is commonly asked for, which helps reproduce NR bugs.

4. RESULTS

In this section, we present the results of our study for each research question.

4.1 Frequency and Comparisons (RQ1)

Table 1 presents the percentage of NR bug reports for each repository. The results of our study show that, on average, 17% of all bug reports are resolved as non-reproducible at least once in their life-cycles.

Figures 2–5 depict the results of comparing NR bug reports with other resolution types. For each bug repository, the NR bug reports are shown with grey background. We

Table 3: NR Categories and Rules.

1) Interbug Dependencies: NR report cannot be reproduced because it has been implicitly fixed:

a) as a result or a side effect of other bug fixes
b) although it is not clear what patch fixed this bug
c) and the bug is a possible duplicate of or closely related to other *fixed* bugs.

Example #759127 in FIREFOX*:* R: ❝*It is now working with Firefox 15.0.1. I believe it was fixed by the patches to #780543 and #788600 [...].*❞

2) Environmental Differences: NR report cannot be reproduced due to different environmental settings such as:

a) cashed data (e.g., cookies), user settings/preferences, builds/profiles, old versions
b) third party software, plugins, add-ons, local firewalls, extensions
c) databases, Virtual Machines (VM), Software Development Kits (SDK), IDE settings
d) hardware(mobile/computer) specifics such as memory, browser, Operating System (OS), compiler
e) network, server configuration, server being down/slow.

Example #261055 in FIREFOX*:* D: ❝*This is probably an extension problem. Uninstall your extensions and see if you can still reproduce these problems.*❞ R: ❝*that did it, I just uninstalled all themes and extensions, and afterwards reinstalled everything from the getextensions website. And now everything works again [...].*❞

3) Insufficient Information: NR report cannot be reproduced due to lack of enough details in the report; developers request more detailed information:

a) regarding test case(s)
b) pertaining to precise steps taken by the reporter leading to the bug
c) regarding different conditions that result in the reported bug.

Example in INDUSTRIAL*:* D: ❝*Cannot reproduce this problem. [...] go to the main screen of the blackberry device, hold ALT and press L+O+G, it will show the logs. That information can help us to some degree.*❞

4) Conflicting Expectations: NR report cannot be reproduced when there exist conflicting expectations of the application's functionality between end-users/developers/testers:

a) misunderstanding of a particular functionality or system behaviour when it works as designed (i.e., lack of documentation)
b) misunderstanding of (non)supported features, out of scope, dropping support or obsolete functionality in newer versions
c) change in requirements
d) misunderstandings turning into QA conversations

Example #29825 in ECLIPSE*:* D: ❝*PDE Schema works as designed [...] Since we cannot tell when you want to use tags and when you want to use reserved chars as-is, you need to escape them yourself EXCEPT, again, when between the '<pre>' and '</pre>' tags that we recognize as a special case [...].*❞

5) Non-deterministic Behaviour: NR report cannot be reproduced deterministically.

Example #MDLSITE-2255 in MOODLE*:* R: ❝*This happened for me again, and then went away again (started working). It seems there is an intermittent problem.*❞

6) Other: NR report cannot be reproduced due to various other reasons, such as mistakes of reporters:

Example #MDL-35391 in MOODLE*:* R: ❝*I'm so sorry... This is not a bug. It occurred because I have been using Moodle 2.3 since beta and overwriting old source in the same directory. Could admin please delete this ticket? Sorry again.*❞ D: ❝*Thanks for the explanation, closing.*❞

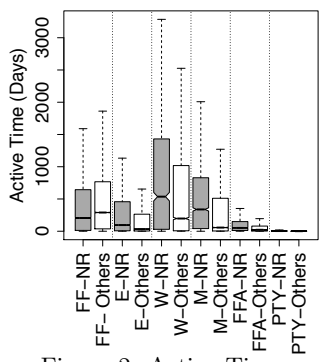

Figure 2: Active Time

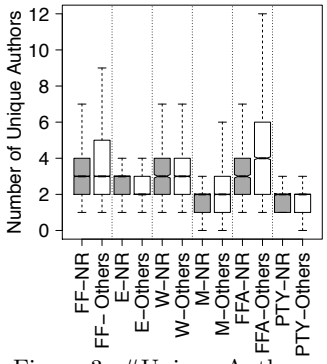

Figure 3: #Unique Authors

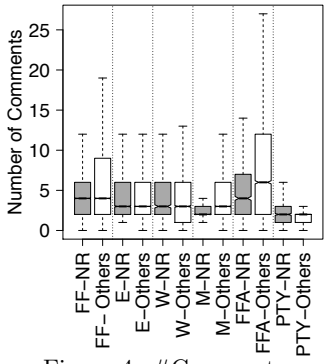

Figure 4: #Comments

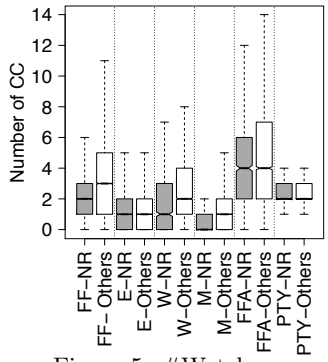

Figure 5: #Watchers

ignore outliers for legibility. Table 4 shows the mean, median, standard deviation, max and p-value (Mann-Whitney) for each comparison metric.[2] The results show that active time is significantly different, i.e., NR bug reports are on average *three months longer active* than non-NR bug reports. For the number of unique authors, comments, and CC/watchers, the results are statistically significant ($p < 0.05$), but the observed differences, having almost the same medians, are not indicative, meaning that NR bug reports receive as much attention from reporters and developers as any other resolution type.

4.2 Cause Categories (RQ2)

Table 3 shows the classification rules we used in our cause category investigation. Figure 6 shows the six main categories that emerged in our analysis, with their overall rate. As shown, "Interbug Dependencies" is the most common category with having 45% of the NR bugs, followed by "Environmental Differences" (24%), "Insufficient Information" (14%), "Conflicting Expectations" (12%), "Non-deterministic Behaviour" (3%) and "Other" (2%). Additionally, Figure 7 depicts the rate of the six cause categories per bug repository. We provide examples of each category below.

Interbug Dependencies. Bug reports in this category are those that cannot be reproduced because they have been indirectly fixed with or without explicit software patches. This category implies that there are bug reports that perhaps are not identical but semantically closely related to each other. Overall, this is the most common cause category we observed in the study (45%). Examples include:

#767543 in FIREFOX: ❝*D: Works for me for Beta 15, Aurora 16, and Nightly 17 with Swype Beta 1.0.3.5809 on Galaxy Nexus. I think my fix for bug #767597 fixed this bug.*❞

#177769 in FIREFOX: ❝*D: Will resolve this as NR since we don't know which checkin fixed this.*❞

#259652 in ECLIPSE: ❝*D: I remember fixing this but can't find the bug. Since it doesn't happen in HEAD, marking as NR.*❞

#723250 in FIREFOX ANDROID: ❝*D: This should be fixed now with my latest changes on inbound. Specifically, bug 728369.*❞

[2]Min was 0 in all cases.

Table 4: Descriptive statistics between NR and Others, for each defined metric: Active Time (AT), # Unique authors (UA), # Comments (C), # Watchers (W), from all repositories.

Metric	Type	Mean	Median	SD	Max	p-value
AT	NR	396	154	553	4534	0.00
	Others	313	40	531	4326	
UA	NR	3.16	3	2.22	85	0.00
	Others	3.06	2	2.61	103	
C	NR	5.14	3	7.9	459	0.03
	Others	5.93	3	12.5	1117	
W	NR	2.1	1	3	159	0.00
	Others	2.7	2	4.3	145	

Environmental Differences. Bug reports in this category cannot be reproduced due to environmental settings that are different for developers/testers/end-users. This category accounts for 24%. Examples include:

#353838 in ECLIPSE: ❝*D: [..] your install got corrupted because of incompatible bundles. You could first try to disable or uninstall Papyrus and if that doesn't help try to remove the Object Teams bundles.*❞

#DTP-01 in INDUSTRIAL: ❝*D: This has something to do with the* XCODE *settings on the build machine. Try to build it on another computer and see if it works. I cannot reproduce this on my iPhone, iPad + simulators.*❞

#456734 in FIREFOX: ❝*R: I solved the problem by uninstalling firefox (without extensions) and installing version 3.0.1 again, and then updating it again to 3.0.2. It's a mystery for me but it helped so it's solved.*❞

Insufficient Information. This is when developers need more specific and detailed information from the reporters. This category accounts for 14% of NR bug reports. Examples of this category include:

#125142 in ECLIPSE:: ❝*D: I haven't been able to reproduce this bug in the Java debugger [...]. Do you have a test case that displays the launch happening in the foreground? marking as NR. Please reopen with a reproducible test case if this is still occurring.*❞

#3103 in MEDIAWIKI: ❝*D: I'm going to resolve this bug (as NR) on the grounds that without further details of the circumstances in which it occurs, there's really not much we can do...* ❞

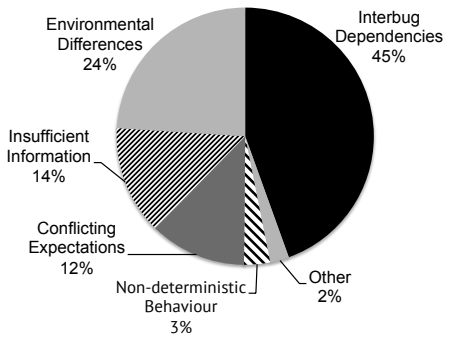

Figure 6: Overall Rate of NR Categories.

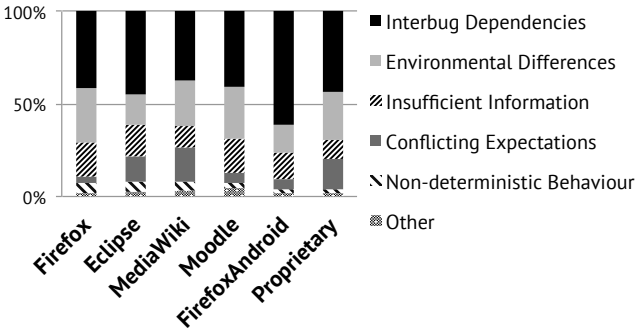

Figure 7: Rate of root cause categories in each bug repository.

#19880 in MediaWiki: ❝*D: I've tested ru.wikipedia.org in IE5.5 on Windows 2000, IE6 on Windows XP, IE7 on Windows Vista, IE8 on Windows Vista. I was unable to reproduce this problem. Perhaps the reporter of this bug could be more specific.*❞

Also tickets are also resolved as NR when there is no response from reporters for several months. For example:

#10014 in MediaWiki: ❝*D: Closing 'support bug' due to lack of response; if the problem persists, please consider taking it up on the mediawiki-l mailing list.*❞

In the Firefox project, an automated message is set up in the bug tracking system, which states ❝*This bug has had no comments for a long time. Statistically, we have found that bug reports that have not been confirmed by a second user after three months are highly unlikely to be the source of a fix to the code. [...] If this bug is not changed in any way in the next two weeks, it will be automatically resolved (NR).*❞

Conflicting Expectations. This category represents bug reports in which there exist conflicting expectations of the software between end-users/developers/testers. Such conflicts could be related to a particular system behaviour, functionality, feature, software support, activity, input/output types and ranges, or specification documentation. In these scenarios the user believes there is a bug in the system since what they see is different from their mental model and/or expectations. As a result, the reported bugs are not really bugs and thus cannot be reproduced by developers. 12% of NR bug reports fall into this category. Some examples are:

#956483 in Firefox Android: ❝*D: [...]* `getDefaultUAString` *is not what you think. That controls the UA of the Java HTTP requests we make in Fennec. This is not used by the Gecko networking and rendering engine. You need to use the normal Gecko preferences to change the UA. This might work: [...] R: Thanks. That works.*❞

#12593 in MediaWiki: ❝*R: [...] I can live with this because it is consistent and predictable behaviour. Thinking about it, it is probably desirable that the system works this way for migration purposes; for example: when importing a dump into a newer* MediaWiki *version.*❞ says the reporter.

#19943 in MediaWiki: ❝*D: Seems ok to me. As long as extensions are passing their path as either a full URL (with protocol) or relative from the docroot they should be fine [...] Checked all extensions in MW SVN that call this, and they all seem to be ok, [...]. Works for me, no real issue with* `addExtensionStyle()` *here. R: Ah, I see. Needs documenting, then [...].*❞

#17265 in MediaWiki: ❝*R: Preferably, the user and talk page of the other username should be deleted, because it'll be impracticable to merge. I hope this will be implemented and will help a lot of people. D: Works for me. There's an extension [...] that does this. Also, there's a maintenance script [...] that can be used for edit attribution, if someone wanted to manually merge two users.*❞

Non-deterministic Behaviour. This category represents bugs that cannot be reproduced deterministically, meaning that the failure is intermittent and triggered at random; and thus difficult to analyze. 3% of NR bug reports are in this category. An example of a developer comment is given below:

#DTP-02 in Industrial: ❝*D: This crash is very random, hard to reproduce. But my guess is it is network/analytics related. It may have to do with the user scrolling through a number of events in the Schedule section which the app cannot keep up with and then eventually crashes.*❞

Other. Any other reason not covered in the other 5 categories would fall under this category (2%). One common instance in this category is bug reports that are mistakenly reported, such as opening an old ticket by mistake, or running the system with incorrect permissions.

#152 in MediaWiki: ❝*R: For the last 3 hours I made the assumption that we could only import articles from the template namespace ... Additionally I made an error in my testing page that I just figured out. Closing...*❞

#8966 in MediaWiki: ❝*R: Shame on me, The function is not broken, I [mis]understood the syntax.*❞

4.3 Common Transition Patterns (RQ3)

68% of NR bug reports are resolved directly from the initial status (New/Open→Resolved(NR)). For the remaining 32%, there are various transition scenarios that NR bugs go through, changing their status and resolution. Table 5 presents some of the observed examples of the *status transitions* of NR bug reports. For instance, the bug report in row #6 changes resolutions 5 times: *Fixed → Fixed → Invalid → NR → Fixed*.

We examined *resolution transitions* of NR bug reports more closely, and plotted a resolution change pattern graph for the six bug repositories, which is depicted in Figure 8. In order to extract a common pattern for all the six repositories,

Table 5: Examples of STATUS (RESOLUTION) transitions of **NR** bug reports.

#	Status (Resolution)
1	NEW→RESOLVED(**NR**)→REOPENED→RESOLVED(**NR**)→REOPENED→RESOLVED(**NR**)
2	NEW→RESOLVED(**NR**)→REOPENED→ASSIGNED→RESOLVED(FIXED)→REOPENED→ RESOLVED(FIXED)
3	NEW→RESOLVED(FIXED)→REOPENED→RESOLVED(WONTFIX)→RESOLVED(**NR**)
4	NEW→RESOLVED(FIXED)→REOPENED→RESOLVED(**NR**)→REOPENED→NEW→RESOLVED(WONTFIX)
5	NEW→RESOLVED(FIXED)→REOPENED→RESOLVED(**NR**)→REOPENED→RESOLVED(FIXED)
6	UNCONFIRMED→NEW→RESOLVED(FIXED)→REOPENED→RESOLVED(FIXED)→REOPENED→RESOLVED(INVALID) →REOPENED →RESOLVED(**NR**)→REOPENED→RESOLVED(FIXED)
7	NEW→ASSIGNED→NEW→RESOLVED(**NR**)→REOPENED→ASSIGNED→RESOLVED(FIXED)→VERIFIED
8	NEW→ASSIGNED→RESOLVED(**NR**)→REOPENED→ASSIGNED→RESOLVED(LATER)→REOPENED→NEW→ASSIGNED →NEW →ASSIGNED→RESOLVED(FIXED)
9	UNCONFIRMED→RESOLVED(INCOMPLETE)→UNCONFIRMED→RESOLVED(INCOMPLETE)→RESOLVED(FIXED) →RESOLVED(**NR**)

we abstracted away custom (repository-specific) resolutions such as *Later*, *Remind*, *Expired*, *Rejected*, *Unresolved*, and *NotEclipse*. The custom resolutions are clustered as *Custom Resolutions* in Figure 8. The other resolutions shown in the graph were common in all the repositories.

We distinguish between two types of transitions in Figure 8: the black arrows indicate all the direct connections to the NR resolution, i.e., all the fan-ins and fan-outs; the grey arrows indicate the indirect connections between other resolutions and NR resolution. To avoid cluttering the figure, we only show weights larger than 2% on the graph. As the figure illustrates, 69% of the transitions are resolved as NR from the beginning. 4.6% of the transitions are from *Fixed* to NR. For instance, #376902 in FIREFOX was first resolved as *Fixed* then changed to NR with a comment: ❝*Fixed refers to problems fixed by actual code changes to* FIREFOX*. Here NR is the correct resolution.*❞

Interestingly, 5.1% of the transitions are from NR to *Fixed*. We explore fixed NR bug reports further in the following subsection.

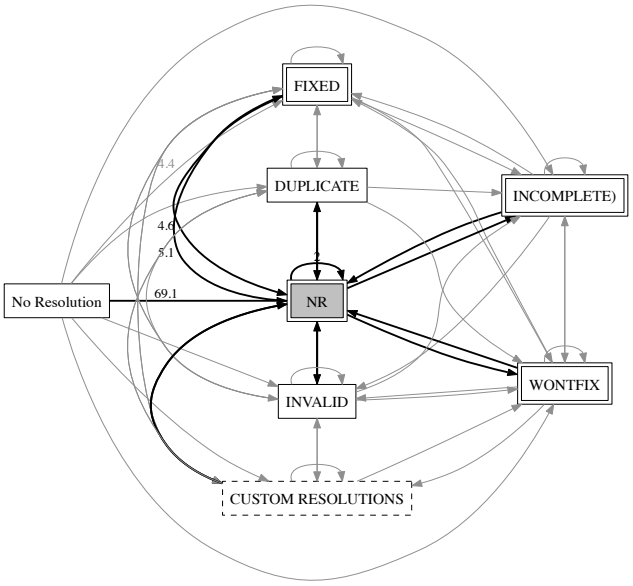

Figure 8: Resolution-to-Resolution Transition Patterns of NR Bug Reports. To avoid cluttering the figure, only weights larger than 2% are shown on the graph.

4.4 Fixed Non-reproducible Bugs (RQ4)

The last column in Table 1 shows that, on average, 3% of all NR bugs become *Fixed*. From these, around 66% actually become reproducible as valid bugs and are fixed with code patches. They mainly fall into "Insufficient Information", "Environmental Differences", and "Conflicting Expectations" cause categories. Some examples include:

#209834 in ECLIPSE: ❝*D: Now, when you described the problem more precisely I realized it's a valid bug. I checked it in both 3.3.1.1 (which you're using) and N20071221-0010 (on which I'm on at this moment) and I can see the problem by clicking the 'Apply' button several times - a resource matched to *.a rule changes it's state even though the rule is enabled all the time. I'll put up a fix in a minute [...]*❞ ("Insufficient Information" category)

#533470 in FIREFOX: ❝*R: [...] I think I got to the bottom of it. The confusion was caused by kernel settings: I thought it was fixed, but actually it was just a ipv6 module getting automatically loaded. The problem still exists when there is no kernel ipv6 support available. I've submitted a simple patch to pulseaudio which will hopefully be accepted and solve the problem.*❞ ("Environmental Differences" category)

#245584 in FIREFOX: ❝*D: the problem was because NS_NewURI was failing - perhaps it was failing because there was something about the URL from IE's data that our networking system couldn't handle? Since this was particular to that URL in that person's set of typed URLs in IE, it didn't show up for everyone [...]*❞ ("Environmental Differences" category)

Interestingly, there were no code patches assigned to the rest of *Fixed* NR bug reports (34%). These are mislabelled reports, as the *Fixed* resolution is used when ❝*a fix for a bug is checked into the tree and tested*❞ [1]. From these, around 24% are in the "Interbug Dependencies" category. For example:

#705166 in FIREFOX: ❝*D1: [...], guess this bug is fixed in the latest nightly. Working fine for me too. D2: [...] WorksForMe is not a correct resolution for this one. The bug was actually fixed by the patch in bug 704575.*❞

5. DISCUSSION

In this section, we discuss our general findings related to non-reproducible bug reports and discuss some of the threats to validity of our results.

5.1 Quantitative Analysis of NR Bug Reports

Our investigation in the quantitative attributes of NR and other types of bug reports shows that NR bug reports are as costly and important as the rest since they receive the same

amount of attention as other bug report types, in terms of the number of comments and developers involved. Developers are typically reluctant to close these bug reports, and they try to involve more people and ask questions through comments. As a result, NR bug reports remain open substantially — around three months on average— longer than other types of bug reports. This clearly points to the uncertainly and low level of confidence developers have when dealing with NR bugs. Possible explanations for leaving NR bug report open longer could be that (1) they do not want to be responsible in case the NR bug turns out to be a real (reproducible) bug that needs fixing, (2) they hope more concrete information will be provided to help reproduce the bug, and/or (3) they wait for someone else to be assigned to the report who knows how to reproduce the bug.

5.2 Fixing NR Bugs

As our results from the six repositories have shown, on average 17% of all bug reports are resolved as NR. Among those, 3% are later marked as *Fixed*. A deeper investigation into the *Fixed* NR reports revealed that around 66% of them become indeed reproducible and fixed with code patches. The rest (34%) have no code patches assigned to them, from which around 24% are in the "Interbug Dependencies" category. This means overall only 1.98% of all NR bug reports are fixed with an explicit code patch. This indicates that the majority of NR bug reports remain unreproducible.

5.3 Interbug Dependencies

On the other hand, 45% of all NR bugs were categorized as "Interbug Dependencies", where they were non-reproducible because they were implicitly fixed in other bug reports. Therefore, we expected the percentage of the explicit fixed NR bugs to be higher than 3%. However, it turns out that developers use the NR resolution for reports that are resolved as a consequence of other bug fixes. This implies that almost half of all NR bugs are actually (implicitly) fixed bugs. We believe coming up with automatic solutions that would cluster these interbug dependent reports based on inferred historical characteristics would help the developers in this regard.

5.4 Mislabelling

Our findings indicate that many reports are misclassified. These misclassifications happen not due to human errors but also because of the fact that the available resolutions in the repositories do not cover all possible scenarios. For instance, many developers use the NR (or WorksForMe) resolution when they actually mean the bug report is irrelevant, unimportant, or even fixed. This is different than the formal definition of NR bugs (see Section 2). We observed many inconsistencies and ambiguities around the usage of the *Fixed* and *NR* resolutions, in particular in cases where a bug report needs to be marked as "fixed with no code patches". Bugs 376902 and 705166 (subsections 4.3 and 4.4, respectively) are examples of these cases.

5.5 Different Domains and Environments

The active time of NR bug reports in the INDUSTRIAL repository is much lower than the open source repositories (see Figure 2). According to Table 1, NR bugs are more prevalent in the studied open-source projects, i.e., they pertain to 11–28% in the open source repositories and 4% in the industrial case. In addition, as presented in the last column of Table 1, although the rate of NR bug reports is lower in the INDUSTRIAL case, the rate of fixed NR bug reports is higher, compared to the open source repositories. Although these findings apply to our sample repositories, possible reasons behind these differences could be that in commercial projects, there is more at stake and, therefore, developers (1) spend more time and effort in reproducing even hard to reproduce bugs, and (2) cannot afford to simply ignore NR bugs. It could also be that the company has a brute force policy in terms of closing bug reports as soon as possible. On the other side, developers in open source projects have less time to spend and less urgency to fix/close a bug report.

Additionally, in the mined repositories, the rate of NR bug reports in desktop applications is more than web and mobile applications, i.e., they are in the range of 13–28% for desktop, 11–12% for web, and 4–15% for mobile applications. Figure 2 indicates that NR bug reports have a lower active time in the repositories of the mobile applications, compared to the desktop and web applications. In addition, the difference between the medians of NR and other bug reports is the highest in the web applications, followed by the desktop, and mobile applications in our study.

5.6 Communication Issues

The two categories "Insufficient Information" (14%) and "Conflicting Expectations" (12%) indicate that there is a source of uncertainty and lack of proper communication between the reporters and resolvers. Herzig et al. [23] observed this uncertainty as a source of misclassification patterns in their recent bug report study. Equipping bug tracking systems with better collaboration tools would facilitate and enhance the communication needs between the two parties. For the category "Environmental Differences" (24%), techniques that make it easier to capture the steps leading to the bug through, e.g., record/replay methods [22], monitoring the dynamic execution of applications [14], or capturing user interactions [27] would be helpful to reproduce the bug report.

5.7 Threats to Validity

Our manual classification of the bug reports could be a source of internal threats to validity. In order to mitigate errors and possibilities of bias, we performed our manual classification in two phases where (1) the inference of rules was initially done by the first author; the rules were cross validated and uncertainties were resolved through extensive discussions and refinements between the first two authors; the generated categories were discussed and refined by all the three authors, (2) the actual distribution of bug reports into the 6 inferred categories was subsequently conducted by the first author following the classification rules inferred in the first step.

In addition, since this is the first study classifying NR bug reports, we had to infer new classification rules and categories. Thus, one might argue that our NR rules and categories are subjective with blurry edges and boundaries. By following a systematic approach and triangulation we tried to mitigate this threat. Another threat in our study is the selection and use of these bug repositories as the main source of data. However, we tried to mitigate this threat by selecting various large repositories and randomly selecting NR bug reports for analysis.

In terms of external threats, we tried our best to choose bug repositories from a representative sample of popular and actively developed applications in three different domains (desktop, web, and mobile). With respect to bug tracking systems, JIRA and BUGZILLA are well-known popular systems, although bug reports in projects using other bug tracking systems could behave differently. Thus, regarding a degree of generalizability, replication of such studies within different domains and environments (in particular for industrial cases) would help to generalize the results and create a larger body of knowledge.

All repositories except the INDUSTRIAL case are publicly available, making the quantitative findings of our study reproducible.

6. RELATED WORK

We categorize related work into two classes: empirical bug report studies and failure reproduction studies.

Empirical Bug Report Studies. Empirical bug report studies have so far focused on different perspectives including understanding the quality of bug reports [15, 16, 24, 25], reassignments [21], bug report misclassifications [23], reopenings [31, 29], prediction and statistical models [26, 28, 19, 20], bug fixing and code reviewing process [30], and coordination patterns and activities around the bug fixing process [13].

Herzig et al. [23] recently reported that every third 'bug report' is not really a bug report. In a manual examination of more than 7,000 bug reports of five open-source projects, they found 33.8% of all bug reports to be misclassified - that is, rather than referring to a code fix, they resulted in a new feature, an update to documentation, or an internal refactoring. This misclassification introduces errors in bug prediction models: on average, 39% of files marked as defective actually never had a bug. They estimated the impact of this misclassification on earlier studies and recommended manual data validation for future studies. The results of our study also confirm this finding.

Aranda et al. [13] report on a field study of coordination activities around bug fixing, through a combination of case study and a survey of software professionals. They found that the histories of even simple bugs are strongly dependent on social, organizational, and technical knowledge, which cannot be solely extracted through automation of electronic repositories, and that such automation provides incomplete and often erroneous accounts of coordination.

Zimmermann et al. [31] characterized how bug reports are reopened, by using the Microsoft Windows operating system project as a case study, using a mixed-methods approach. They categorized the reasons for reopening based on a survey of 358 Microsoft employees and ran a quantitative study of Windows bug reports, focusing on factors related to bug report edits and relationships between people involved in handling the bug. They propose statistical models to describe the impact of various metrics on reopening bugs ranging from the reputation of the opener to how the bug was found.

Guo et al. [21] present a quantitative and qualitative analysis of the bug reassignment process in the Microsoft Windows Vista project. They quantify social interactions in terms of both useful and harmful reassignments. They list five reasons for reassignments: finding the root cause, determining ownership, poor bug report quality, hard to determine proper fix, and workload balancing. Based on their study, they propose recommendations for the design of more socially-aware bug tracking systems.

To the best of our knowledge, our work is the first to report a characterization study on non-reproducible bug reports.

Failure Reproduction Studies. Apart from the empirical bug studies, there have been a number of studies [22, 14, 27] analyzing and proposing solutions for failure reproduction. Roehm et al. [27] present an approach to monitor interactions between users and their applications selectively at a high level of abstraction, which enables developers to analyze user interaction traces. Herbold et al. [22] use a record/replay approach and monitor messages between GUI objects. Such messages are triggered by user interactions such as mouse clicks or key presses. We believe these techniques can help make NR bug reports easier to understand and reproduce. In this paper, however, we perform a mining study of NR bug reports to understand their nature, leaving possible solutions for future work.

7. CONCLUSION

Working on non-reproducible bug reports is notoriously frustrating and time consuming for developers. In this paper, we presented the first empirical study on the frequency, nature, and root cause categories of non-reproducible bug reports. We mined 6 bug tracking repositories from three different domains, and found that 17% of all bug reports are resolved as non-reproducible at least once in their lifecycles. Non-reproducible bug reports, on average, remain active around three months longer than other resolution types while they are treated similarly in terms of the extent to which they are discussed or the number of developers involved. In addition, our analysis of resolution transitions in non-reproducible bug repots revealed that such reports change their resolutions many times. Furthermore, around 2% of all NR bug reports are eventually fixed with code patches, while around half are implicitly 'fixed'.

Our manual examination revealed 6 common root cause categories. Our classification indicated that "Interbug Dependencies" forms the most common category (45%), followed by "Environmental Differences" (24%), "Insufficient Information" (14%), "Conflicting Expectations" (12%), and "Non-deterministic Behaviour" (3%).

Our study shows that many NR bug reports are mislabelled pointing to the need for bug repository systems and developers to resolve inconstancies in the usage of the *Fixed* and *NR* resolutions.

For future work, we plan to focus on (1) bug reports in the "Interbug Dependencies" category to design techniques that would facilitate identifying, linking, and clustering them upfront so that developers would not have to waste time on them, (2) incorporating better collaboration tools into bug tracking systems to facilitate better communication between different stakeholders to address the problem with the other NR categories.

8. ACKNOWLEDGMENTS

This work was supported in part by NSERC, UBC (4YF), and Swiss National Science Foundation (PBTIP2145663).

9. REFERENCES

[1] Bugzilla. http://www.bugzilla.org/docs/.

[2] Bugzilla: Eclipse Bug #106396. https://bugs.eclipse.org/bugs/show_bug.cgi?id=106396.

[3] Bugzilla@Mozilla. https://bugzilla.mozilla.org.

[4] Eclipse Bugzilla. https://bugs.eclipse.org/bugs/.

[5] JIRA. https://confluence.atlassian.com/display/JIRA050/JIRA+Documentation.

[6] MediaWiki Bugzilla. https://bugzilla.wikimedia.org/.

[7] Moodle Tracker! https://tracker.moodle.org/issues/?jql=.

[8] Non-reproducible bug report analyser and empirical data. https://github.com/saltlab/NR-bug-analyzer.

[9] Works on my machine - How to fix non-reproducible bugs? http://stackoverflow.com/questions/1102716/works-on-my-machine-how-to-fix-non-reproducible-bugs.

[10] Bug fields. https://bugzilla.mozilla.org/page.cgi?id=fields.html, Sep 2013.

[11] What is an issue? https://confluence.atlassian.com/display/JIRA/What+is+an+Issue, Sep 2013.

[12] "works on my machine" - how to fix non-reproducible bugs? http://stackoverflow.com/q/1102716, Sep 2013.

[13] J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 298–308. IEEE Computer Society, 2009.

[14] J. Bell, N. Sarda, and G. Kaiser. Chronicler: Lightweight recording to reproduce field failures. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 362–371. IEEE Press, 2013.

[15] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 308–318. ACM, 2008.

[16] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru. An empirical analysis of bug reports and bug fixing in open source Android apps. In *Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR)*, pages 133–143. IEEE Computer Society, 2013.

[17] CNET. Researcher posts Facebook bug report to Mark Zuckerberg's wall, 2013. http://news.cnet.com/8301-1023_3-57599043-93/researcher-posts-facebook-bug-report-to-mark-zuckerbergs-wall/.

[18] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage Publications, Incorporated, 2013.

[19] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall. Method-level bug prediction. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 171–180. ACM, 2012.

[20] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 495–504. ACM, 2010.

[21] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. 'not my bug!' and other reasons for software bug report reassignments. In *Proceedings of the Conference on Computer Supported Cooperative Work*, CSCW, pages 395–404. ACM, 2011.

[22] S. Herbold, J. Grabowski, S. Waack, and U. Bünting. Improved bug reporting and reproduction through non-intrusive gui usage monitoring and automated replaying. In *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '11, pages 232–241. IEEE Computer Society, 2011.

[23] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 392–401. IEEE Computer Society, 2013.

[24] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 34–43. ACM, 2007.

[25] A. Kumar Maji, K. Hao, S. Sultana, and S. Bagchi. Characterizing failures in mobile OSes: A case study with Android and Symbian. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, pages 249–258. IEEE Computer Society, 2010.

[26] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead Jr. Does bug prediction support human developers? findings from a Google case study. In *Proceedings of the International Conference on Software Engineering*, ICSE, pages 372–381. IEEE Computer Society, 2013.

[27] T. Roehm, N. Gurbanova, B. Bruegge, C. Joubert, and W. Maalej. Monitoring user interactions for supporting failure reproduction. In *International Conference on Program Comprehension (ICPC)*, pages 73–82. IEEE, 2013.

[28] H. Seo and S. Kim. Predicting recurring crash stacks. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pages 180–189. ACM, 2012.

[29] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K.-i. Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 249–258. IEEE Computer Society, 2010.

[30] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram. How do fixes become bugs? In *Proceedings of ESEC/FSE*, pages 26–36. ACM, 2011.

[31] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 1074–1083. IEEE Computer Society, 2012.