Empir Software Eng (2018) 23:1704–1742
https://doi.org/10.1007/s10664-017-9559-4

# Are tweets useful in the bug fixing process? An empirical study on Firefox and Chrome

**Mariam El Mezouar[1] · Feng Zhang[1] · Ying Zou[2]**

Published online: 9 November 2017


**Abstract** When encountering an issue, technical users (e.g., developers) usually file the issue report to the issue tracking systems. But non-technical end-users are more likely to express their opinions on social network platforms, such as Twitter. For software systems (e.g., Firefox and Chrome) that have a high exposure to millions of non-technical end-users, it is important to monitor and solve issues observed by a large user base. The widely used micro-blogging site (i.e., Twitter) has millions of active users. Therefore, it can provide instant feedback on products to the developers. In this paper, we investigate whether social networks (i.e., Twitter) can improve the bug fixing process by analyzing the short messages posted by end-users on Twitter (i.e., tweets). We propose an approach to remove noisy tweets, and map the remaining tweets to bug reports. We conduct an empirical study to investigate the usefulness of Twitter in the bug fixing process. We choose two widely adopted browsers (i.e., Firefox and Chrome) that are also large and rapidly released software systems. We find that issue reports are not treated differently regardless whether users tweet about the issue or not, except that Firefox developers tend to label an issue as more severe if users tweet about it. The feedback from Firefox contributors confirms that the tweets are not currently leveraged in the bug fixing process, due to the challenges associated to discovering bugs through Twitter. Moreover, we observe that many issues are posted on Twitter earlier than on issue tracking systems. More specifically, at least one third of

Communicated by: David Lo

✉ Mariam El Mezouar
mariam@cs.queensu.ca

Feng Zhang
feng@cs.queensu.ca

Ying Zou
ying.zou@queensu.ca

[1] School of Computing, Queen's University, Kingston, ON, Canada

[2] Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada

issues could have been reported to developers 8.2 days and 7.6 days earlier in Firefox and Chrome, respectively. In conclusion, tweets are useful in providing earlier acknowledgment of issues, which developers can potentially use to focus their efforts on the issues impacting a large user-base.

**Keywords** Bug report · Social network · Twitter · Bug fixing

## 1 Introduction

Technical users (e.g., developers) get used to file a bug report into an issue tracking system, when they encounter a software bug. When reporting a bug, technical users are required to adhere to strict guidelines for bug reporting; otherwise, a bug report may be rejected. For example, the bug reporter needs to provide a detailed description of the bug and the concrete steps to reproduce the bug. Hence, a bug report is well structured. Structured bug reports can help the development team to accelerate bug triaging and fixing. However, non-technical end-users can be intimidated by the overhead of filing a well structured bug report. Sometimes, non-technical end-users just want to voice their opinions on more familiar channels where they can draw some attention to their problems. In particular, non-technical end-users may prefer to use social networks.

Social networks are good sources to collect timely reactions from people on hot and ongoing events. It is very important to promptly collect the feedback from end-users, especially after a new software release. Although a new release is expected to fix bugs and add new features, new bugs can also be introduced inadvertently. For example, a flood of complaining tweets forced Apple to immediately pull the release of iOS 8.0.01 (Welch 2014). For a software system, end-users may switch to its competitors, if the development team is not responsive to the end-users' feedback and the reported bugs remain unfixed for a long period of time. To retain the loyalty of the end-users, it is of great interest to study how social networks can be used to improve the bug fixing process.

Another value of collecting feedback from a large user base is that different users experience very diverse real-life scenarios and have varied configurations. Therefore, the development team can better know how their software product runs in every possible setting. We are interested to investigate how the feedback posted by end-users on social networks, and particularly the micro-blogging platform Twitter, could be useful to the bug discovery and fixing process.

Twitter is the largest micro-blogging platform, where end-users post short messages (i.e., at most 140 characters), called tweets, to express opinions or report events. Over the last few years, Twitter has gained a rapid popularity and adoption. As of September 2016, Twitter has 313 million monthly active users. Tweets are free text without a rigorous structure that report instant feedback from end-users. Tweets have been studied heavily using sentiment analysis (Kouloumpis et al. 2011; Pak and Paroubek 2010; Go et al. 2009) and opinion mining (Pak and Paroubek 2010; Liu and Zhang 2012; O'Connor et al. 2010). These studies aim to help product owners answer questions, such as: *how do people feel about our product?* or *what would people want us to add to the product?* We focus on the feedback on software products only.

Particularly, we perform an empirical study to understand the value of using tweets in the bug fixing process. We choose to study Chrome and Firefox, the two most popular web browsers. The two systems are rapidly released (i.e., approximately every six weeks), thus

it is time sensitive to collect feedback promptly for these two systems so that more bugs reported in one release could be resolved in the subsequent release.

Specifically, we investigate the following four research questions:

RQ1. **How accurately can our approach map tweets and bug reports?**
To the best of our knowledge, there is no well-established approach to map tweets and bug reports. In this paper, we propose an automated approach to map tweets and bug reports. The approach is based on Apache Lucene, a widely used engine for general-purpose search. To evaluate the accuracy of our approach, we assess the precision of our approach when considering/not considering different pre-processing steps. Three evaluators are asked to manually check the relevance between each bug report and its mapped tweets. The results show that our approach achieves a precision between 76% and 84%, and outperforms the baseline approach involving no pre-processing steps (i.e., 47% to 52%) with a large margin.

RQ2. **What web browser issues receive more feedback from end-users through Twitter?**
We refer to a bug report as a tweeted bug report, if at least one tweet is successfully mapped to the bug report. Otherwise, we call it a non-tweeted bug report. We compare the topics between the tweeted and non-tweeted bug reports. We find that tweeted bug reports are more likely to contain issues related to performance, security, and audio/video issues in both browsers.

RQ3. **Does the development team handle a bug report differently if the problem is mentioned on Twitter?**
We compare the tweeted and non-tweeted bug reports in terms of the assigned severity and three fine-grained intervals (i.e., delay before response, delay before assignment, and duration of bug fixing). As shown by our experiments, end-users' tweets do not appear to be currently leveraged in the bug fixing process for both systems. No statistical significance is observed in both subject systems except for one case where the tweeted bug reports of Firefox tend to have higher severity. We further reach out to developers from both Firefox and Chrome. The Firefox contributors in charge of the social media support report that bug discovery through Twitter, while possibly useful, could be challenging (no feedback was received from the Chrome developers). We further collect bug-reporting tweets, and manually summarize and submit the bugs reported to the issue tracking system. As a result, the bugs reported are acknowledged by the developers as legitimate bugs; thus highlighting the possible usefulness of the tweets.

RQ4. **Can we use tweets to achieve an early discovery of bugs?**
The findings in RQ3 shed some doubt on the value of using tweets in the bug fixing process. Given the strength of tweets in providing timely feedback from a large user base who have very diverse settings (e.g., machine type, operating system, and machine configuration), we are interested to examine if using tweets can lead to an early discovery of bugs. Indeed, we find that 33.4% and 33.5% of the bugs could have been reported earlier to developers averagely by 8.2 days and 7.6 days in Firefox and Chrome, respectively. With a well designed tool for summarizing tweets, the development team could discover bugs earlier, and possibly focus their efforts on the bugs that affect a large user base.

**Paper Organization** We present the overview of our case study design in Section 2, and describe details in Sections 3, 4, 5, and 6. We report the results of our experiments and

present our findings in Section 7. We discuss the threats to validity and implications in Sections 8 and 9. The related work is described in Section 10. We conclude our work in Section 11.

## 2 Overview of Our Case Study

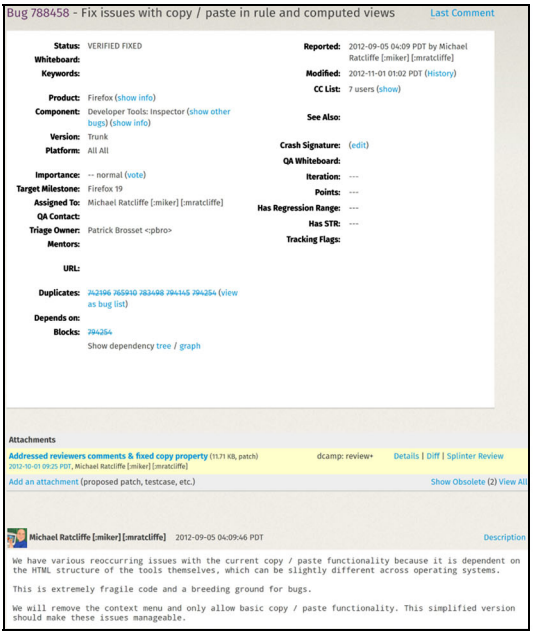
In this section, we describe the overview of our case study design. The design of our study is depicted in Fig. 2.

First, we download the bug reports from issue tracking systems (e.g., Bugzilla) (see Section 4). Then, we crawl tweets from Twitter (see Section 5). We pre-process both the bug reports and the tweets, by cleaning and normalizing the collected data. We develop an automated tool to collect, pre-process, and select the relevant feedback from users on Twitter.

Second, we map the bug reports and the tweets using a text-similarity off-the-shelf search engine (see Section 6). Figure 1 shows an example of a possible mapping between a bug report (Fig. 1a) and a tweet (Fig. 1c). Our approach is evaluated by three non-author evaluators (i.e., RQ1).

Third, we perform an empirical study to understand if the bug reports that are successfully mapped to users' tweets are different from the bug reports without associated tweets, in terms of the bug fixing intervals, the severity/priority levels, and the types of reported bugs. As such, we compare the bug fixing process between the tweeted and non-tweeted bug reports in RQ2 and RQ3. We further get in touch with developers from both Firefox and Chrome to verify the findings of our quantitative study (Fig. 2).

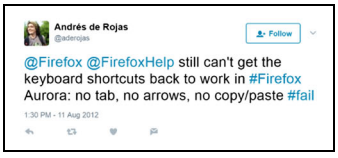
Finally, we investigate whether acknowledging the feedback from end-users on Twitter could help the development team discover bugs at an earlier stage (i.e., RQ4).




**(a)** An example report (Bugzilla #788458)

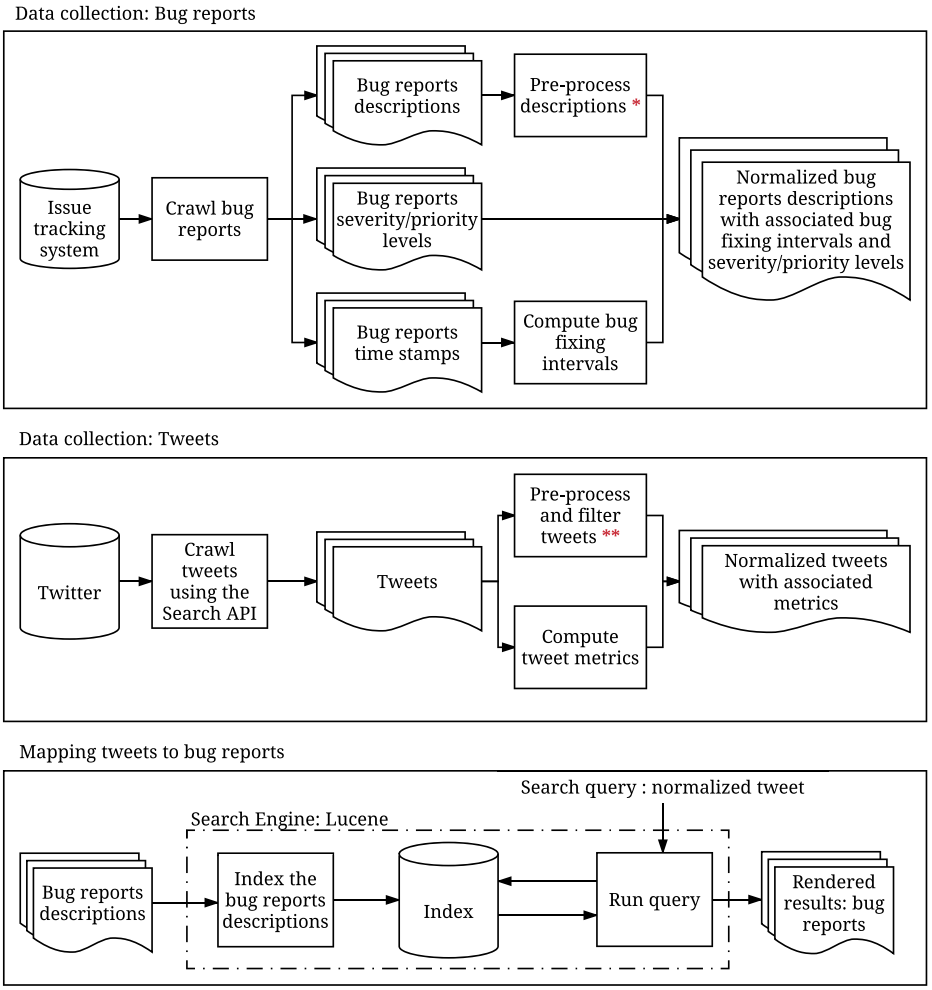

**(b)** An example of the 1st matched tweet


**(c)** An example of the 2nd matched tweet




**(d)** An example of the 3rd matched tweet

**Fig. 1** An example of a Firefox bug report and three tweets possibly related to the bug




* Details on the pre-processing steps of the bug reports are provided in Section 4.3

** Details on the pre-processing and filtering steps of the tweets are provided in Section 5.2

**Fig. 2** Overview of the case study design

## 3 Subject Systems

The social networks, especially Twitter, provide timely end-users' feedback on hot and ongoing events. A new release of a software product may not be a trending event. Therefore, a new release may not receive many tweets, unless the product has a large user base. As such, we choose to study the two most widely[1] used browsers (i.e., Chrome and Firefox) that are released on a regular basis (i.e., approximately every six weeks). In particular,

[1]According to W3Schools' browser statistics in March 2016, Chrome ranks the first with 69.9% of the usage share of browsers, followed by Firefox that has approximately 17.8% of worldwide usage share of the browsers. URL: http://www.w3schools.com/browsers/browsers_stats.asp

Firefox and Chrome have adopted the rapid release cycle since June 2011 (Version 5.0) and October 2011 (Version 15.0), respectively. The rapid release cycle allows the collection of more frequent feedback (i.e., The developers are likely to hear from the end-users every six weeks).

Firefox is an open source browser developed by Mozilla Foundation and opens its issue tracking system to the public. However, Chrome is proprietary software of Google, and its issue tracking system is unaccessible by the public. As the bug fixing process is recorded in the issue tracking system, we turn to the issue tracking system of Chromium which is the open-source version of Chrome. Chromium and Chrome largely share the same code and features, except for some minor differences in features and licenses. We further observe that Chromium has only developmental releases while all official stable releases belong to Chrome. Therefore, issue reports of Chrome can be easily distinguished from those of Chromium using the release version. Specifically, we select issue reports that are associated to the stable releases of Chrome only.

In total, we study 37 consecutive versions of Firefox (from Version 5.0 to Version 41.0), and 34 stable releases of Chrome (from Version 15.0 to Version 48.0). The dates of the Firefox releases can be obtained from the *history of Firefox*[2] Wikipedia page. The dates of the Chrome releases can be found on the *Google Chrome release history*[3] Wikipedia page. The detailed dates are presented in Table 1.

## 4 Bug Reports

In this section, we first describe the background of the bug fixing process, then explain our collection and pre-processing of the bug reports.

### 4.1 Bug Fixing Process

Bug reports record the entire bug fixing process. The typical life cycle of the bug fixing process starts from the moment a bug is reported and ends when it is closed. A detailed life cycle of Firefox bug reports is described in the work by Weiss et al. (2007). Details of the life cycle of Chrome bug reports can be found on the *Bug Life Cycle and Reporting Guidelines* web page.[4] Although slightly different terminologies are used in different issue tracking systems, the life cycle of a bug report usually goes through the phases shown in Fig. 3. Details of each state are described as follows:

1) `NEW`: A bug report status is initially set to `NEW` when it is first created. For each new bug report, the development team initiates bug triaging to find the most appropriate developer to work on the bug. Bug triaging may require several iterations until the most appropriate developer is identified.
2) `ASSIGNED`: After triaging, a developer is assigned to work on resolving the bug. The status is changed to `ASSIGNED`. Developers start to inspect and fix the bug.
3) `RESOLVED`: Once the bug is fixed, the status is set to `RESOLVED`. Usually, developers will submit a patch and related test cases. At this stage, the testing phase starts.

[2] https://en.wikipedia.org/wiki/History_of_Firefox

[3] https://en.wikipedia.org/wiki/Google_Chrome_release_history

[4] https://www.chromium.org/for-testers/bug-reporting-guidelines

**Table 1** Descriptive statistics of the studied releases of Firefox and Chrome

| Firefox | | | | Chrome | | | |
|---|---|---|---|---|---|---|---|
| Version | Date | # Bugs | # Tweets | Version | Date | # Bugs | # Tweets |
| 5.0 | 2011-06-21 | 389 | 2890 | 15.0 | 2011-10-25 | 86 | 629 |
| 6.0 | 2011-08-16 | 345 | 1744 | 16.0 | 2011-12-13 | 205 | 623 |
| 7.0 | 2011-09-27 | 73 | 2126 | 17.0 | 2012-02-08 | 200 | 590 |
| 8.0 | 2011-11-08 | 282 | 1531 | 18.0 | 2012-03-28 | 197 | 556 |
| 9.0 | 2011-12-20 | 265 | 1364 | 19.0 | 2012-05-15 | 78 | 527 |
| 10.0 | 2012-01-31 | 353 | 817 | 20.0 | 2012-06-26 | 72 | 780 |
| 11.0 | 2012-03-13 | 236 | 956 | 21.0 | 2012-07-31 | 71 | 878 |
| 12.0 | 2012-04-24 | 284 | 953 | 22.0 | 2012-09-25 | 58 | 759 |
| 13.0 | 2012-06-05 | 199 | 788 | 23.0 | 2012-11-06 | 92 | 902 |
| 14.0 | 2012-06-26 | 243 | 3205 | 24.0 | 2013-01-10 | 69 | 658 |
| 15.0 | 2012-08-28 | 233 | 1483 | 25.0 | 2013-02-21 | 169 | 815 |
| 16.0 | 2012-10-09 | 238 | 1263 | 26.0 | 2013-03-26 | 914 | 1020 |
| 17.0 | 2012-11-20 | 262 | 1842 | 27.0 | 2013-05-21 | 703 | 925 |
| 18.0 | 2013-01-08 | 240 | 1049 | 28.0 | 2013-06-17 | 746 | 798 |
| 19.0 | 2013-02-19 | 350 | 1123 | 29.0 | 2013-08-20 | 711 | 1174 |
| 20.0 | 2013-04-02 | 329 | 2406 | 30.0 | 2013-09-18 | 640 | 1201 |
| 21.0 | 2013-05-14 | 217 | 1567 | 31.0 | 2013-11-12 | 830 | 1310 |
| 22.0 | 2013-06-25 | 245 | 1835 | 32.0 | 2014-01-14 | 564 | 1231 |
| 23.0 | 2013-08-06 | 287 | 2362 | 33.0 | 2014-02-18 | 834 | 1195 |
| 24.0 | 2013-09-17 | 198 | 1725 | 34.0 | 2014-04-02 | 679 | 1154 |
| 25.0 | 2013-10-29 | 219 | 1549 | 35.0 | 2014-05-20 | 807 | 1336 |
| 26.0 | 2013-12-10 | 278 | 1381 | 36.0 | 2014-07-15 | 579 | 910 |
| 27.0 | 2014-02-04 | 230 | 1706 | 37.0 | 2014-08-26 | 687 | 1354 |
| 28.0 | 2014-03-18 | 226 | 1929 | 38.0 | 2014-10-07 | 617 | 958 |
| 29.0 | 2014-04-29 | 402 | 3102 | 39.0 | 2014-11-12 | 763 | 1530 |
| 30.0 | 2014-06-10 | 284 | 1100 | 40.0 | 2015-01-20 | 579 | 1073 |
| 31.0 | 2014-07-22 | 209 | 1122 | 41.0 | 2015-03-03 | 656 | 1055 |
| 32.0 | 2014-09-02 | 236 | 958 | 42.0 | 2015-04-14 | 452 | 1103 |
| 33.0 | 2014-10-14 | 325 | 3423 | 43.0 | 2015-05-19 | 708 | 1538 |
| 34.0 | 2014-12-01 | 248 | 1705 | 44.0 | 2015-07-21 | 477 | 1037 |
| 35.0 | 2015-01-13 | 329 | 1878 | 45.0 | 2015-09-01 | 436 | 1188 |
| 36.0 | 2015-02-24 | 303 | 1830 | 46.0 | 2015-10-13 | 422 | 1078 |
| 37.0 | 2015-03-31 | 381 | 1628 | 47.0 | 2015-12-01 | 310 | 1250 |
| 38.0 | 2015-05-12 | 369 | 2144 | 48.0 | 2016-01-20 | 272 | 891 |
| 39.0 | 2015-07-02 | 231 | 1649 | | | | |
| 40.0 | 2015-08-11 | 378 | 1294 | | | | |
| 41.0 | 2015-09-22 | 249 | 858 | | | | |

4) `VERIFIED`: If the submitted bug fix passes testing, the tester marks the bug report as `VERIFIED`. The bug fix is ready for release.
5) `CLOSED`: The bug is marked as `CLOSED`, which indicates the end of the fixing process. However, a bug report can be reopened if the corresponding fix is unsatisfactory. In this case, the status is set to `REOPEN` and bug triaging is initiated again.

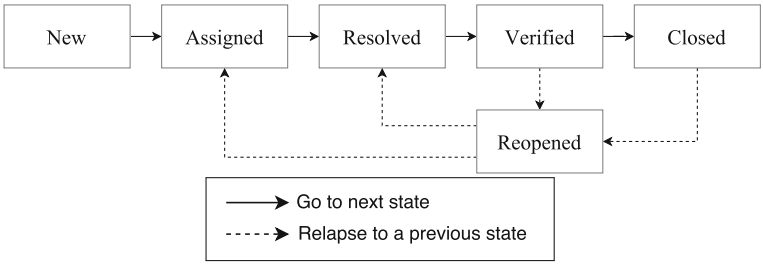



**Fig. 3** Life cycle of a bug report

In order to assess the impact that tweets may have on the bug fixing process, we compute fine-grained intervals during the entire bug fixing process. Similar to the work by Zhang et al. (2012), we identify and compute four fine-grained intervals: delay before reported ($DBR_{\text{eported}}$), delay before response ($DBR$), delay before assignment ($DBA$), and duration of bug fixing ($DBF$). To compute these intervals, we mine timestamps of all changes made on the status of every bug report. Specifically, we compute each interval in the following way (also shown in Fig. 4).

1) Delay Before Reported ($DBR_{\text{eported}}$) measures how long it takes the development team to acknowledge a bug after a new release. It is computed as the interval between the date of a version release (i.e., $T_{\text{release}}$) and the date at which a bug report is filed (i.e., $T_{\text{reported}}$). The creation date is recorded in the bug report. However, it is not clear when the bug was introduced. Considering that both Firefox and Chrome are set to auto-update by default on end-users' machines, we assume each reported bug was introduced by the most recent release, or is a dormant bug revealed by the most recent release. In both cases, the bug is only a nuisance to the users after it is triggered by the most recent release. As aforementioned, we extract the date of the most recent release from release history of these two browsers. Then, we compute the interval $DBR_{\text{eported}}$ using the following equation:

$$DBR_{\text{eported}} = T_{\text{reported}} - T_{\text{release}} \tag{1}$$

2) Delay Before Response ($DBR$) reflects the delay of the development team on taking initial actions after a bug is reported. It is calculated as the interval between the filing of a report (i.e., $T_{\text{reported}}$) and its first response from the development team. The first response can be captured by several types of initial actions taken by the development team, which are the addition of a developer to the Carbon Copy (i.e., CC) list of the bug (i.e., the list of developers who receive emails about any updates to the bug report), the change in the status of the bug, or the posting of the first comment. We mine the timestamp of these actions and choose the timestamp of the earliest action as the time

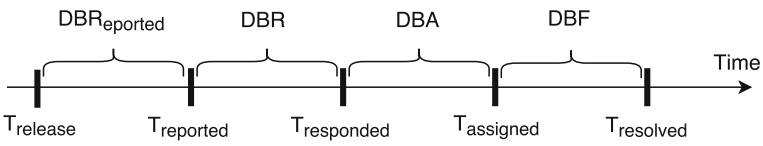



**Fig. 4** Intervals of the bug fixing process

of the first response (i.e., $T_{\text{responded}}$). Then, we calculate the interval $DBR$ using the following equation:

$$DBR = T_{\text{responded}} - T_{\text{reported}} \quad (2)$$

3 Delay Before Assignment ($DBA$) captures the time spent in triaging during the bug fixing process. It is computed as the interval between the first response to a bug and its assignment to a developer. As bug triaging may require several iterations, we choose the most recent assignment as the final assignment. We mine the timestamp of the most recent assignment as $T_{\text{assigned}}$. Then, we compute the interval $DBA$ using the following equation:

$$DBA = T_{\text{assigned}} - T_{\text{reported}} \quad (3)$$

4) Duration of Bug Fixing ($DBF$) describes the duration spent on inspecting the bug and performing the corresponding code changes by the assigned developer. It is calculated as the interval between the assignment of the bug and the resolution of the bug ($T_{\text{resolved}}$) when the status of the bug is changed to `RESOLVED`. $DBF$ does not capture multiple resolve actions, but reflects the duration of the successful and final resolution. We calculate the interval $DBF$ using the following equation:

$$DBF = T_{\text{resolved}} - T_{\text{assigned}} \quad (4)$$

### 4.2 Bug Reports Collection

Bug reports are stored in issue tracking systems. Firefox uses Bugzilla[5] as its issue tracking system. From Firefox Bugzilla, we crawl 14,489 fixed bug reports associated to the releases under study. The median number of bug reports for a single release is 262 (see Table 1). Chrome uses the Chromium tracker[6] as its issue tracking system. Although the Chromium tracker contains bug reports for both Chrome and Chromium, they can be distinguished from each other using the *ReleaseBlock* field. The *ReleaseBlock* field can be either *stable* or *developmental*. As aforementioned, bug reports for stable releases belong to Chrome. Therefore, we download all bug reports with the *ReleaseBlock* field set to *stable*. In total, we crawl 15,771 fixed bug reports associated with the stable Chrome releases under study. The median number of Chrome bug reports for a single release is 477 (see Table 1).

Bug reports from both Bugzilla and Chromium tracker share a similar structure with a different flavor. For instance, both issue tracking systems record the description of the problem and the severity or priority of the problem. A full list of historical activities for each bug report is recorded separately from its main page in Bugzilla. In the Chromium tracker, a new comment is posted on the main page of the bug report when the status is updated. We extract both descriptive information and the history information from all the bug reports for the two subject systems.

### 4.3 Bug Reports Pre-Processing

A manual investigation of the bug reports reveals that some bug reports contain logs that are automatically generated for debugging purposes (i.e., automatically generated logs). For example, as shown in Fig. 5, the text in the automatically generated logs contains mainly timestamps and other technical details describing system events. Considering the limitation

[5] https://www.bugzilla.org/

[6] https://bugs.chromium.org/p/chromium/issues/list

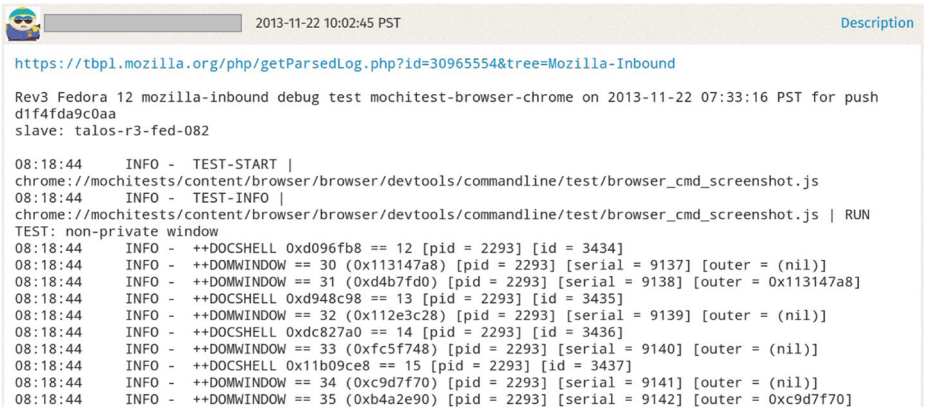



**Fig. 5** Example of automatically generated logs in the description of a bug report

on the number of characters in a tweet, we assume that it is unlikely the end-users would copy and paste the automatically generated logs to Twitter. Therefore, we exclude the lines identified as automatically generated logs from the bug report description prior to mapping the bug reports to the tweets. Bacchelli et al. (2012) propose a sophisticated approach to classify the content of mailing lists at the line level into text, junk, code, patch, and stack trace. For our work, we opt for a simpler approach tailored to the bug reports, where we classify the lines into 2 classes: text and automatically generated logs.

To identify the lines containing automatically generated logs, we parse the bug reports line by line. We observe that multiple instances of automatically generated logs start with a timestamp of the form `hh:mm:ss`. We use the following regular expression to flag such lines: `^\d{2}[:]+\d{2}[:]+\d{2}[ \t]`. In other instances, we identify the automatically generated logs by looking for lines starting with a list of words such as `stack trace`, `slave`, `test-pass`, or `test-unexpected-fail`. We use the following regular expression to identify such patterns `(^stack\strace)|((^slave))|((^test-pass))|((^test-unexpected-fail))`.

Normalization is a process that converts a list of words into a more uniform sequence. The purpose of normalization is to prepare the text for further processing. We perform the normalization process using the following three steps: *1)* removing the non-English words using the Moby words list[7] (the largest list of English words in the world); *2)* removing the English stop words[8] (e.g., about, such, or too); and *3)* reducing all words to their roots, i.e., stemming using the Porter stemmer (Porter 1980). We perform the normalization on all bug reports.

## 5 Tweets

Twitter[9] is the largest micro-blogging service for social networking. End-users can freely post messages (i.e., *tweets*), with a 140-character limitation for each message.

[7] http://icon.shef.ac.uk/Moby/mwords.html

[8] https://github.com/mariamelm2/TwitterPaperRep/blob/master/Data/stop_words.txt

[9] https://twitter.com/

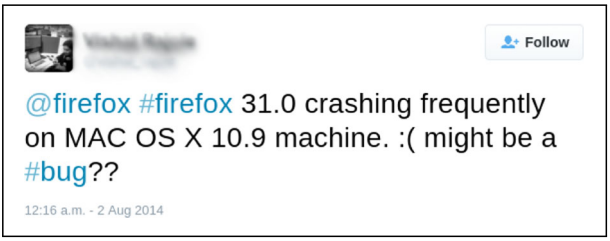



**Fig. 6** Example of a tweet addressed to Firefox using the `"@"` symbol

We assume that the tweets posted around the time when a new version is released are mostly about the new release. This is because both Firefox and Chrome are set to auto-update by default on end-users' machines. As such, most end-users are very likely to run the new version of each browser soon after the release.

### 5.1 Tweets crawling

It has been reported that the velocity of tweet posting is around 6,000 tweets per second.[10] Prior to using tweets in the bug fixing process, the first challenge is to identify the subset of relevant tweets from a flood of tweets from Twitter.

**1) Reduce the Search Space** Millions of tweets are posted on Twitter everyday. We aim to only retrieve the tweets posted by end-users to report bugs encountered in the subject systems. Therefore, we consider two features of Twitter to reduce the search space of tweets related to Chrome and Firefox. One feature of Twitter is the hashtag (i.e., `"#"`), which is a type of label or metadata tag that is used to group and for end-users to find messages related to a specific topic or theme. End-users can create and use hashtags by placing the hash character `"#"` in the text of the tweet. Another feature of Twitter is the usage of the `"@"` symbol. A tweet can be addressed to a specific end-user using the `"@"` symbol followed by the username of the target end-user (e.g., `@firefox`). Figure 6 shows an example of a tweet addressed to Firefox using both the `"@"` symbol and the hashtag `"#"`.

Both Firefox and Chrome maintain their official Twitter accounts. The Firefox official account has 2.86 million followers and over 27,000 tweets, as of March 2016. The Firefox account is used to communicate with the Firefox end-users and announce updates about the browser. The Chrome official account has 5.9 million followers and over 1,000 tweets. The Chrome account is mostly used for the promotion of the new features of the browser.

In this study, we identify the tweets addressed to each official account using the `"@"` symbol. We do not use the hashtag (`"#"`) to identify tweets about Firefox or Chrome. Although tweets containing the hashtags `"#Chrome"` or `"#Firefox"` might relate to issues about Chrome and Firefox, using these hashtags result in a large amount of false positives. The two first authors manually investigate a statistically significant sample of tweets (361 tweets) containing the hashtags `"#Chrome"` or `"#Firefox"`, to have an estimation of the number of false positives. We find that 60.8% of the tweets are false positives. For example, a false positive is the following: "*Resolved the DCC bug for #Firefox. Still working on #Chrome. Common code, different behavior. #Javascript*". Another example is as follows: "*Google reports a "high severity" bug in IE/Edge, no patch available*

[10] http://www.internetlivestats.com/twitter-statistics

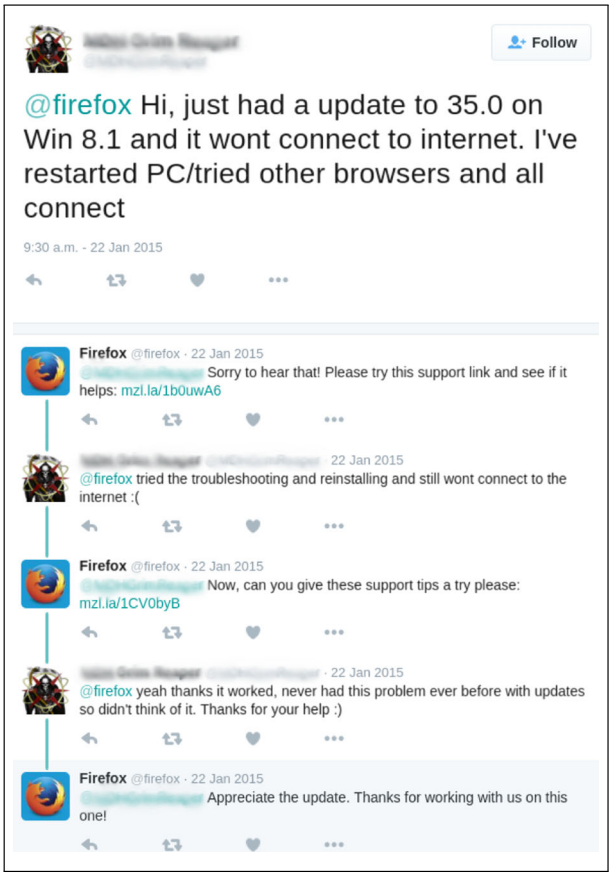



**Fig. 7** Example of a conversation on Twitter between an end-user and the Firefox official account (Usernames are blurred to preserve privacy)

*#chrome #firefox #opera*". Within the remaining 39.2% of the tweets that are true positives, less than 10% are bug-reporting tweets. It is hard to automatically identify whether the problem included in the tweet is about one browser or the other just by parsing the words in the tweet. Therefore, it is challenging to automatically distinguish false positives from true positives. In this paper, we want to focus on studying the possible use of the end-users' feedback (i.e., the tweets) on the bug fixing process. Therefore, we map tweets to the bug reports. To improve the accuracy of the mapping, we leave the overhead of filtering tweets with the hashtags `"#Chrome"` or `"#Firefox"` to a future study. On the other hand, the `"@"` symbol ensures that a tweet is directly addressed to the official account of a specific browser.

**2) Perform the Search** Twitter provides a search API that allows queries against the indices of recent or popular tweets, such as the `"until"` operator that is used to retrieve tweets posted before a given date. An example of a query used to retrieve the tweets associated with Firefox Version 40 is "lang:*en* to:*@firefox* since:$d_{start}$ until:$d_{end}$", where $d_{start}$ is the release date of Version 40, and $d_{end}$ is the day right before the next release.

Figure 7 depicts an example of a Twitter conversation. The main tweet, shown in a bigger font, has 5 replies and is the first tweet in the conversation. For each tweet, we further

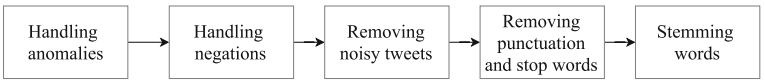



**Fig. 8** Tweet processing steps

retrieve the conversation in which the tweet appears. A tweet can be either an original tweet or a reply to another tweet. For each tweet, we extract the ancestors and the replies, if any.

We categorize the tweets by releases, based on the dates they were posted. The crawled tweets span over four years. The number of tweets posted after each release of Firefox and Chrome are presented in Table 1.

## 5.2 Tweets Pre-Processing

The tweets are subjects to the use of abbreviations and to the occurrence of misspellings. It is common that many words are misspelled or abbreviated. To address this issue, we first handle the anomalies (i.e., abbreviations and misspellings). Moreover, the tweets retrieved are not all relevant to bugs, as some of them merely express opinions or ask general questions. An example of an irrelevant tweet is: "*@firefox can't believe that the worlds best browser is almost 10!*". Therefore, we further filter irrelevant tweets.

The two steps are depicted in Fig. 8, as well as the normalization steps of the words in the tweets. We provide the details of each step in the subsequent paragraphs.

**1) Correct Anomalies** Anomalies are the misspellings, abbreviations and non-standard orthography that exist in the tweets. To correct the anomalies, we collect a list of common abbreviations used by users on Twitter and replace them with the actual correct words. The list of abbreviations is gathered from various online sources.[11,12,13,14,15] Examples of common non-standard orthography in Twitter are shown in Table 2. Furthermore, we examine a statistically representative sample of tweets to identify some of the most common misspellings in the tweets and correct them before further processing. The sample has 361 tweets that are randomly selected from the entire population, using a 95% confidence level with a 5% interval.[16] Few examples of the most common Twitter misspellings are shown in Table 3.

**2) Filter out Irrelevant Tweets** The existence of bug-related terms, such as "lag" or "crash", in a tweet possibly indicates that the tweet reports a bug. We build a customized dictionary of bug-related terms by manually reviewing the aforementioned statistically representative sample of tweets. Examples of bug-related terms that we obtain are *issue*, *bug*, *problem*, *fail*, *freeze*, and *glitch*. A similar approach is adopted by Villarroel et al. (2016) to define a dictionary of bug-related terms, when they identify user reviews of mobile apps that report bugs.

[11] http://www.socialmediatoday.com/content/top-twitter-abbreviations-you-need-know

[12] http://www.webopedia.com/quick_ref/Twitter_Dictionary_Guide.asp

[13] http://www.noslang.com/twitterslang.php,searchcrm.techtarget.com/definition/Twitter-chat-and-text-messaging-abbreviations

[14] http://marketing.wtwhmedia.com/30-must-know-twitter-abbreviations-and-acronyms/

[15] https://digiphile.wordpress.com/2009/06/11/top-50-twitter-acronyms-abbreviations-and-initialisms/

[16] www.surveysystem.com/sscalc.htm

**Table 2** Examples of common twitter abbreviations or acronyms

| Twitter abbreviations and accronyms | Meaning |
|---|---|
| ICYMI | In case you missed it |
| MTF | More to follow |
| Twaffic | Twitter traffic |
| TY | Thank you |
| TT | Trending topic |
| RTQ | Read the question |
| ... | |

Nevertheless, tweets containing negated bug-related terms (e.g., "no lag" or "does not crash"), such as the one shown in Fig. 9, can be misleading when identifying the bug-reporting tweets. Villarroel et al. (2016) address a similar challenge in user reviews by identifying the negated terms using the Stanford parser (Socher et al. 2013) and discarding them. We adopt the same approach and remove the bug-related terms that are negated. The tweet shown in Fig. 9 thus becomes "*@firefox 3.6.1 , Its Super fast, And I liked The Mac os x Skin.*". Therefore, it is discarded from the set of relevant tweets.

To detect the instances of negated bug-related terms, we use Part-Of-Speech tagging from the Stanford parser (Socher et al. 2013). We use one of the available interfaces of the Stanford parser API.[17] For instance, the sentence "the browser is not laggy" is processed into the following parse tree. The meanings of the parser tags (e.g., NP = Noun Phrase) can be found in Bies et al. (1995).

```
(ROOT
  (S
    (NP (DT the) (NN browser))
    (VP (VBZ is) (RB not)
      (ADJP (JJ laggy))
    )
  )
)
```

The parser also returns a list of dependencies (i.e., grammatical relationships among words) as follows. The meanings of the possible dependencies (e.g., neg = negation modifier) are available in the Stanford parser user manual.[18] The number following each word in the list of dependencies (e.g., laggy-**5**) refers to the sequence of the word in the phrase.

```
root ( ROOT-0 , laggy-5 )
det ( browser-2 , The-1 )
nsubj ( laggy-5 , browser-2 )
cop ( laggy-5 , is-3 )
neg ( laggy-5 , not-4 )
```

We then use the "neg" label (i.e., the last item in the list of dependencies above) to identify the negated bug-related term "laggy".

[17] http://projects.csail.mit.edu/spatial/Stanford_Parser

[18] https://nlp.stanford.edu/software/dependencies_manual.pdf

**Table 3** Examples of common twitter misspellings

| Twitter common misspellings | Correct spellings |
|---|---|
| dont | do not |
| da | the |
| B | be |
| youre | you are |
| its | it is |
| cant | cannot |
| ... | |

Finally, we obtain 6,044 tweets that are identified as reporting Firefox bugs (out of 54,293 tweets that are originally retrieved) and 6,158 bug-reporting tweets for Chrome (out of 38,349 tweets that are originally retrieved).

**3) Normalize Tweets** Similar to bug reports, we normalize the text of the tweets. In particular, we remove punctuation (e.g., {, . !}) and special symbols (e.g., {& # $}) from the collection of tweets. Then we remove the non-English words using the Moby words list to retain only the words that are correctly spelled and we exclude all stop words. Finally, we stem the words using the Porter stemmer.

### 5.3 Tweets-Related Metrics

A bug-reporting tweet that receives higher attention from other users might indicate how critical and popular the problem is among the users. A tweet receives higher attention if it has been liked and shared by many other users. We propose to use the following five metrics to assess the popularity of tweets:

1) **# Favorites** measures the agreement and interest of other users in the content of a tweet. It is defined as the number of times that a tweet is marked as a favorite (i.e., liked by a user);
2) **# Retweets** is a means to measure how much the information in the tweet has been diffused among Twitter users. It is defined as the number of times that a tweet has been retweeted (i.e., re-posted by other users for their followers to see);
3) **Has replies?** is a means to assess the relevance of a tweet through the acknowledgment by the software provider. It is defined as whether the tweet has received a reply from the official Twitter account of the browser. Such replies are usually used to answer a question or acknowledge a problem faced by a user (An example is shown in Fig. 10);

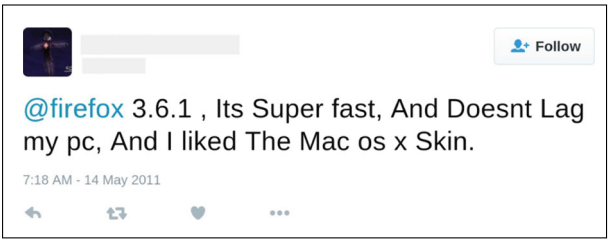



**Fig. 9** Example of a tweet containing a negated bug-related term


**Fig. 10** Example of a tweet that received a reply from the official Firefox account (Usernames are blurred to preserve privacy)

4) **Duration** reflects the relevance of the tweet over time among the Twitter users. It is computed as the total duration of the conversation that a tweet is part of (i.e., the list of replies to the tweet). An example of a Twitter conversation is shown in Fig. 7;
5) **Interaction interval** measures the engagement of Twitter users with the tweet, through the velocity of their interactions. It is defined by computing how fast users interact with each others in a conversation. It is possible to have sub-threads within a Twitter conversation. We do not solve the non-trivial threading problem in this case (i.e., identifying the sub-threads within a Twitter conversation). The interaction interval is measured by *a)* identifying all the tweets that are marked as a reply to the initial tweet X; *b)* computing the time difference between each two consecutive tweets in the conversation; and *c)* reporting the average of the time differences.

## 6 Mapping Tweets to Bug Reports

To understand whether the users' feedback from tweets is currently leveraged in the bug fixing process, we need to establish the links between bug-reporting tweets and bug reports. With the mapping between tweets and bug reports, we can distinguish the bug reports whose issues are reported by Twitter users from the bug reports that are not mentioned on Twitter. Then, we compare the bug fixing process of the two groups of bug reports, in order to understand if the tweets are used in the bug fixing process in the current practice.

When mapping bug reports and tweets, we aim to find pairs of bug reports and tweets that describe a similar problem. Therefore, we compute the text similarity between bug reports and tweets for the mapping. However, a tweet may contain uninformative words that are irrelevant to the described problem. For example, in the following tweet "*Do you*

*know if anyone else is having issues w/ Firefox 16 on Lion with copy and paste shortcuts? Paste works, but copy doesn't.*", we consider words like "*Do you know if anyone else is*" as uninformative. The uninformative words can inflate the similarity score. Therefore, it is important to extract key words that can capture the main point of a given tweet. Then, we use the extracted keywords to represent the corresponding tweet, and take the extracted keywords as a query to search against all bug reports using an off-the-shelf search engine.

In the following subsections, we provide more details about our approach for mapping tweets to bug reports.

### 6.1 Tweet Keywords Extraction

In this study, we extract the $n$-grams of each tweet as keywords to describe its main point. Unlike bag of words, $n$-grams are sequences of $n$ relevant words that do not appear consecutively by accident (e.g., *neural network* or *call cell phone*).

First, we remove the stop words from the tweets since the stop words do not contribute to describe the main topic of a tweet. Then, we extract the full list of n-grams from the entire collection of bug-reporting tweets using the Natural Language Toolkit[19] (NLTK) (Bird 2006). NLTK is a suite of program modules written in Python to support research in natural language processing and computational linguistics. We choose NLTK because it is a well established NLP tool that has been used in numerous prior research studies (MacMahon et al. 2006; Piwowar 2011; Hill et al. 2016; Yao and Van Durme 2014).

After the full list of n-grams is extracted from the entire collection of bug-reporting tweets, we identify the n-gram(s) that appear in each tweet as its keywords. A given n-gram could appear in one or more tweet. For example, the tweet "*@firefox After some testing, it seems that firebug is the cause, since it is causing javascript to hang frequently*" has two matched bi-grams: "*firebug cause*'" and "*javascript hang*". Therefore, we use the two bi-grams as keywords to represent this tweet.

The extraction of $n$-grams has one parameter $n$, which is the number of words in a sequence. It can be any positive integer that is greater than one. However, increasing $n$ decreases the number of extracted $n$-grams, which leads to a lower chance of successfully matching tweets to $n$-grams. Tweets that do not contain $n$-grams can not be mapped to bug reports. To choose an appropriate $n$ for this particular task, we compute the percentage of tweets that can be assigned $n$-grams with varying values of $n$. With $n = 2$ (a.k.a. bi-grams), we can assign n-grams to approximately 87% of the 6,044 subject tweets related to Firefox, and 90.3% of the 6,158 subject tweets related to Chrome. When setting $n = 3$ and $n = 4$, the percentages of tweets that can be assigned $n$-grams are respectively 78% and 65% of the 6,044 subject tweets related to Firefox. The percentage of tweets that can be assigned $n$-grams drops as we increase $n$. Besides, we observe that the words that are captured by the $n$-grams of sizes 3 and 4 are always already captured by the $n$-grams of size 2. For example, from the tweet: "*@firefox Version 37.0.2, no updates available. The problem sites give me: Secure Connection Failed. Same issue on clean install as well*", we obtain the following bi-grams: "*clean install, connect fail, connect secur, fail issu, problem site, problem updat, updat version*", the following tri-gram: "connect fail secur", and no four-gram. Therefore, we choose $n = 2$ (i.e., bi-grams extraction) in our study.

[19] http://www.nltk.org/

### 6.2 Tweet Keywords Querying

To identify the bug reports that are related to a given tweet, we use the extracted keywords (i.e., bi-grams) as a query to search against all bug reports. We choose Lucene as our search engine, since it has been used in a prior study on bug reports (Weiss et al. 2007), and has proved to work well in high profile platforms, such as Twitter, LinkedIn, and Jira (Picorini 2015). Lucene[20] is a free and open source library for information retrieval.

For each release of each subject system, we build a Lucene database using all the bug reports belonging to the same release. Then, for each tweet, we use the extracted keywords (i.e., bi-grams) as a query in Lucene and retrieve a list of bug reports. Tweets are also assigned to specific releases based on the dates they were posted. Only tweets and bug reports belonging to the same release can be mapped. The retrieved bug reports are ordered by the score of similarity to the query.

The Lucene similarity score is implemented as a variant of TF-IDF. In a nutshell, the similarity score varies with the number of times the query terms (i.e., bi-grams of a tweet) occur in a document (i.e., a bug report), and inversely with the number of times the query terms occur in all documents (i.e., corpus of bug reports). Other factors are also factored in the Lucene similarity score, such as $coord(query, document)$ (i.e., how many of the query terms are found in the document), and $lengthNorm(t, d)$ (i.e., a measure of the importance of a term according to the total number of terms in the field). More details on the components of the Lucene similarity score can be found on the similarity Java class webpage.[21] At present, there is no absolute calibration for the highest score returned. It is difficult to determine from the similarity scores the level of relevance between a query and the matched document (Rowe 2013). Therefore, we can not set a unique threshold to determine if a query matches a document. Alternatively, we choose top-$K$ retrieved bug reports for each query as its matched bug reports. The smaller $K$ is, the more strict the mapping is. We perform a sensitivity analysis by varying $K \in \{5, 10, 15\}$. We decide not to go beyond 15, since most users of search engines can retrieve a relevant result within the first 15 results returned by a search engine (Schwartz 2014). Otherwise, users would have changed their query. For instance, an average of 71.3% of searches on Google resulted in a click in the 10 first results. In particular, the first 5 results account for 67.6% of all the clicks (Schwartz 2014).

## 7 Results

In this section, we present the results of our study with respect to four research questions.

**RQ1. How Accurately can our Approach Map Tweets and Bug Reports?**

**Motivation** Analyzing tweets has the potential to benefit the bug fixing process. For instance, it is likely to improve end-users' satisfaction if the development team fixes the bugs that are tweeted by many end-users as soon as possible. Mining problems reported in tweets can also help the development team catch missed bugs. However, the potential benefits of analyzing tweets rely on establishing accurate links between tweets and bug reports.

[20] http://lucene.apache.org/core/5_3_1/

[21] https://lucene.apache.org/core/5_3_1/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

In this question, we aim to evaluate the accuracy of our approach for mapping tweets and bug reports. Our evaluation provides a basis for further approaches on mapping tweets and bug reports.

**Approach** To evaluate the accuracy of our approach, we perform a manual analysis in two steps:

1. We assess the mapping precision resulting from the different Lucene configurations (number of results $K \in \{5, 10, 15\}$). We accordingly select the appropriate number of results $K$ to be used in the remainder of the study, to maximize precision.
2. We evaluate the different preprocessing steps listed in Section 4.3 for bug reports and in Section 5.2 for tweets. We assess the precision gain resulting from each preprocessing step, following a similar approach as Villarroel et al. (2016).

To conduct the manual analysis, we collect a sample of mapped bug reports and associated tweets. We randomly select a sample of 361 bug reports from each subject system. The sampled bug reports can statistically represent the population with a 95% ± 5% confidence.[22] For each selected bug report, we identify all the tweets for which the bug report is returned as a match by Lucene.

We ask three non-author evaluators to assess the mappings of the sampled bug reports resulting from our final approach and the multiple baselines (i.e., considering/not considering the different preprocessing steps). The three evaluators are all graduate students majoring in computer science. For each selected bug report, the evaluators are presented with the possibly matching tweets. The evaluators are asked to mark the tweets they believe are a possible match to the bug report. All evaluators work independently, and they are not told whether a baseline approach or our approach is being evaluated. To assess the agreement among evaluators, we use Fleiss' kappa (Fleiss 1971). Fleiss' kappa is a statistic to measure agreement among two or more evaluators for categorical items (i.e., whether or not a pair of <bug report, tweet> is mapped correctly). Larger Fleiss' kappa means more agreement among evaluators, with a maximum value of 1 indicating complete agreement.

We compute the precision to measure the accuracy of the mapping between tweets and bug reports. Specifically, precision is calculated as the proportion of correctly mapped tweets relative to the total number of mapped tweets for each bug report. Higher precision values indicate better performance of the mapping. Given that we have a large amount of bug reports and tweets (e.g., Firefox has 14,489 bug reports and 6,044 tweets), an exhaustive manual evaluation is required to identify all the pairs <bug report, tweet> that should be mapped and to compute the recall. Moreover, it is a very common practice to use precision to evaluate information retrieval tasks (Buckley and Voorhees 2000). Therefore, we do not use recall but only precision for the evaluation.

**Findings Setting the parameter $K$ to 5 in Lucene returns the most acceptable precision values.** As described in Section 6.2, our approach has a parameter $K$ that controls the number of retrieved bug reports for each query. The sensitivity analysis using $K \in \{5, 10, 15\}$ demonstrates that at least 47.8% of bug reports can be mapped to tweets. Specifically, when the number of retrieved bug reports is set to 5 (i.e., $K = 5$), tweets can be mapped to 47.8% and 50.8% of bug reports in Firefox and Chrome, respectively. When increasing $K$

[22] www.surveysystem.com/sscalc.htm

**Table 4** Coverage of bug reports that are mapped to tweets with varying configuration (i.e., $K \in \{5, 10, 15\}$) of our mapping approach

| Browser | Firefox | | | Chrome | | |
|---|---|---|---|---|---|---|
| Configuration of $K$ (# of retrieved bug reports) | 5 | 10 | 15 | 5 | 10 | 15 |
| Coverage (% of mapped bug reports) | 47.8% | 65.2% | 73.5% | 50.8% | 69.7% | 79.8% |

to 10, tweets are mapped to more bug reports (i.e., 65.2% in Firefox and 69.7% in Chrome). Increasing $K$ can always result in more bug reports that are mapped to tweets with lower text similarity scores. Detailed coverage of bug reports is reported in Table 4.

The purpose of this analysis is to select a search strategy that achieves an acceptable precision/recall trade-off. Increasing $K$ can always result in more bug reports that are mapped to tweets with lower text similarity scores (i.e., higher recall and lower precision). However, the focus of this work is to study the differences between the bug reports associated to tweets and the bugs reports with no associated tweets. As such, it is more critical in this case to obtain the true links, rather than the totality of the links. The precision of the mapping drops for $K$ values larger than 5, as expected. The average drop in precision shown in Table 5 (i.e., -14% for $K = 10$ and -20% for $K = 15$) is significant and might constitute a threat to the validity of the mapping. Therefore, we use the $K = 5$ Lucene configuration in the remainder of the study.

**Identifying the bug-reporting tweets and extracting n-grams from tweets result in the highest precision gain of the mapping approach.** We can observe the impact of each preprocessing step conducted prior to the Lucene mapping in Table 6 when: (i) removing noise from the bug reports (+3% precision gain); (ii) correcting anomalies in tweets (+4% precision gain); (iii) identifying the bug-reporting tweets (+11% precision gain); (iv) normalizing the bug reports and tweets (+3% precision gain); and (v) extracting n-grams from tweets (+9% precision gain). Our approach achieves a precision ranging from 76% to 84%, while the baseline approach involving no preprocessing step yields a precision from 47% to 52%. We compute the Fleiss' kappa among the three evaluators. The value of the Fleiss' kappa is 0.72, indicating a substantial agreement, according to Landis and Koch (1977). A paired Mann-Whitney U test is used to test the following alternative hypothesis:

$H_a^1$ : *The precision of the final mapping approach is higher than the precision of the baseline approach.*

The Mann-Whitney U test reveals a significant improvement (i.e., $p$-value is always less than 2.2e-16) in terms of precision between the baseline approach with no text processing and the final mapping approach. Better precision is achieved, thanks to the n-gram analysis and the identification of the bug-reporting tweets. Indeed, not all the tweets describe a bug, and usually only some words in a tweet are relevant to describing a bug.

**Table 5** Average precision of the mapping approach based on three different Lucene configurations ($K \in \{5, 10, 15\}$) and based on the results from three evaluators

| | $K = 5$ | $K = 10$ | $K = 15$ |
|---|---|---|---|
| Evaluator 1 | 76% | 61% | 58% |
| Evaluator 2 | 84% | 71% | 63% |
| Evaluator 3 | 78% | 63% | 59% |
| **Average** | **79%** | **65%** | **59%** |

**Table 6** Average precision of the mapping approach tweets resulting from multiple baselines

| | No text pre-processing | Remove noise from BRs | Correct anomalies in tweets | Identify bug-reporting tweets | Normalize BRs & tweets | Extract n-grams |
|---|---|---|---|---|---|---|
| Evaluator 1 | 47% | 50% | 56% | 64% | 68% | 76% |
| Evaluator 2 | 52% | 54% | 58% | 71% | 75% | 84% |
| Evaluator 3 | 48% | 52% | 54% | 66% | 67% | 78% |
| **Average** | **49%** | **52% (+3%)** | **56% (+4%)** | **67% (+11%)** | **70% (+3%)** | **79% (+9%)** |

Even though the overall precision returned by the mapping approach is acceptable, it still results in some false positives. For the purpose of highlighting some remaining challenges in mapping tweets and bug reports, we include here some examples of false mappings between tweets and bug reports. The tweet "*@Firefox is having some problems playing YouTube videos... Script error upon opening video pages as well.*" is matched to a bug report with the following title: "*YouTube Audio and Video downloader causing tab crash when loading Youtube homepage in E10S mode*". Even though the issue in both the tweet and the bug report is related to YouTube, the specificities of the issue are not the same. We find several similar instances in the set of mapped tweets and bug reports. The bug reports are longer in length and contain more details. Therefore, even if the terms in a tweet match parts of the bug report, there might be additional details in the bug report that describe details not included in the tweet, or not matching the content of the tweet at all. Additionally, we observe that the bug reports that contain more technical content (e.g., references to code elements) are generally mismatched to the tweets. An example of a bug report that falls under this category is titled: "*Download indicator toolbar button depends on an odd XBL quirk*". The description of similar bug reports mostly contains references to code elements and objects, and does not follow the traditional structure (i.e., reproducible steps, expected results, and actual results). Therefore, similar instances result in false mappings using our approach. Lastly, we observe that the filtering of the bug-reporting tweets has limitations and results in some tweets to be wrongly selected as relevant. For instance, the following tweet "@firefox you claim bug 987323 is fixed on versions 33 and up, but I am having that issue on version 33.1.1. What is going on?" does contain the bug terminology. However, it does not describe what the bug is about, and therefore results in false mappings to the bug reports.

*The precision of our mapping approach varies from 76% and 84%, which outperforms the baseline approach with a large margin. Extracting and using keywords from a tweet can effectively represent the tweet compared to using its original form. The precision is also improved by identifying the bug-reporting tweets.*

### RQ2. What Web Browser Issues Receive more Feedback from End-Users Through Twitter?

**Motivation** End-users and the development team may view a software product from different perspectives. End-users may be particularly interested in some issues based on their

perception of the products, while the development team may focus on some other issues based on the results of the test plans. The mismatch of the focus might affect the satisfaction of end-users. Therefore, it is interesting to identify what issues are most critical to the end-users in the context of web browsers.

**Approach** To address this question, we first extract the topics of bug reports, and then compare the topics between bug reports with associated tweets and bug reports without associated tweets.

Topics are extracted using Latent Dirichlet Allocation (LDA) (Blei et al. 2003), which has been successfully used to identify software concerns (Baldi et al. 2008) and analyze bug reports (Somasundaram and Murphy 2012; Linstead and Baldi 2009). LDA has two major parameters: the number of topics ($N_{\text{topics}}$) and the number of words ($N_{\text{words}}$). Each document (i.e., bug report) is assigned a vector of probabilities (length is $N_{\text{topics}}$) that describe the chance of each topic to appear in the document. Each topic is described by a collection of words (size is $N_{\text{words}}$).

To set an appropriate number of LDA topics ($N_{\text{topics}}$), we use an R package called *Ldatuning*.[23] *Ldatuning* reports the results of four metrics, Arun2010 (Arun et al. 2010), CaoJuan2009 (Cao et al. 2009), Devaud2014 (Deveaud et al. 2014), and Griffiths2004 (Griffiths and Steyvers 2004). The optimal number of topics is decided with a majority vote of the metrics. Based on the results of the *ldatuning* package, we set the number of topics to 200 (i.e., $N_{\text{topics}} = 200$). We further set the number of topic words to 20 (i.e., $N_{\text{words}} = 20$) to ease the understanding of each topic. To ensure that extracted topics are comparable across the two subject systems, we perform a single run of LDA on the corpus of bug reports from both Firefox and Chrome.

To comprehend the extracted topics, we ask three people to manually label and categorize the topics. The three people include two graduate students in computer science who also contributed to the manual analysis conducted in RQ1, and one software engineer with two years of experience in software development. None of them authors this paper. Each topic is labeled based on the list of 20 topic words. For example, the label *unresponsiveness of the browser* is assigned to the topic with the following 20 keywords: "*hang, freez, stop, firefox, respond, continu, forc, minut, unrespons, quit, wait, complet, time, frozen, kill, happen, respons, recov, exit, sudden*" (all are stemmed). In case of disagreements among the evaluators, a short discussion among the evaluators is conducted to reach a consensus. If no consensus is reached, the label agreed on by the majority is used.

To provide a high-level view of the extracted topics, the topics are further categorized based on the browser feature that they are closest to by the authors. One of the evaluators is asked to review the resulting categorization, which we then refine based on the received feedback. Table 7 shows all ten categories of topics and the labels of topics within each category. For instance, the topics *speed of opening a new page* and *memory usage* are assigned to the category *performance*, as both topics describe issues regarding to the performance of the browser. We exclude from the list of resulting topics the following categories of topics: *a)* the 8 topics that could not be labeled (e.g., *"firefox, command, state, theme, agent, expect, interfac, gnome, broken, fedora, notic, wrong, user, persona, regular, exist, skin, aero, area, classic"*); and *b)* the 70 topics that contain only bug reports related keywords (e.g., *"result, actual, step, agent, expect, gecko, build, user, reproduc, window, mozilla, id"*). This results in 122 topics out of the 200 originally extracted topics. During the manual labeling of the

[23] https://cran.r-project.org/web/packages/ldatuning

**Table 7** All ten categories of topics and topic labels within each category

| Topic category | Topic labels |
|---|---|
| Audio and video | Audio related - Video and flash |
| Coding and debugging | Test cases - CSS properties - Coding and debugging - Data structures |
| Configuration and updates | Remote configuration - Packages installation - Synchronization of services - Firefox profiles - Client/server interactions - Versions and upgrades - User privileges - Icon customization |
| Functionalities | Browsing modes - Search engine - Cookies and privacy - Printing - Bookmarks - Management of multiple windows - Editor and text manipulation - User support - User accounts - Passwords management - Cache - History |
| GUI appearance | Resolution - Window size- Colors and transparency - Size and positioning of window - Text appearance - Browser themes |
| GUI logic | Correctness of display - Rendering of files - Mouse actions - keyboard actions - Facebook-related - Correctness of actions in browser - Email-related - Tabs actions - Open and click actions - Pictures and images - Files and directories - Navigation - Page loading - Screeshots - Input fields - Zoom action - Buttons - Menus - Visibility of results - Text suggestions - Session restoration - Progress visibility - Error messages - Address bar - Languages related - Input fields - Visibility of progress - Tabs action |
| OS and hardware | Management of processes - Devices - OS related - Windows related - MAC related - Drivers |
| Performance | Browsing modes - Speed of opening a new page - Performance of animations - Starting Firefox - Time performance - Unresponsiveness - Memory usage |
| Security | Web security - Network security |
| Tools and extensions | Firebug - Management of extensions - Libraries - Inspector |

remaining topics, we observe that some topics are duplicated. Therefore, we merge the duplicated topics, and end up with the 79 topics shown in Table 7.

Finally, we examine what web browser issues receive more feedback from end-users through tweets. We divide all bug reports into two groups: one group consists of all the bug reports that have associated tweets and the other group includes all the remaining bug reports. To study which topics are more appealing to end-users, we compare the probability of each topic category to appear in bug reports between these two groups. We define the null hypothesis as:

$H_0^2$: *there is no difference in the probability of bug reports to have a particular topic category between bugs with and without associated tweets.*

To test the null hypothesis, we apply the Fisher's exact test (Sheskin 2007) with the 95% confidence level. If there is statistical significance (i.e., $p$-value $< 0.05$), we reject the null hypothesis. We further compute the Odds Ratio (OR) (Sheskin 2007) to determine if the corresponding topic category has a higher or lower likelihood to appear in the bug reports that have associated tweets (i.e., bugs posted by end-users).

**Findings End-users are more interested in web browser issues related to performance, security and audio/video in both subject systems**. Specifically, performance issues have 2.01 times higher chance to appear in bugs with associated tweets than in bugs without associated tweets in Firefox. Similar finding (i.e., 2.44 times for performance

**Table 8** Odds ratio and the corresponding $p$-value of the Fishers' exact test on the appearance of topic categories

| | Firefox | | Chrome | |
|---|---|---|---|---|
| | Odds ratio (OR) | $p$-value | Odds ratio (OR) | $p$-value |
| **Audio/video related** | **3.08** | **2.08e-05** | **2.64** | **3.98e-05** |
| Coding and debugging | 0.70 | n.s | **0.53** | **4.35e-02** |
| Configuration and updates | 1.56 | n.s | 0.97 | n.s |
| Functionalities | 1.35 | n.s | 1.13 | n.s |
| **GUI appearance** | **0.42** | **6.55e-05** | **0.55** | **3.38e-04** |
| GUI logic | 0.98 | n.s | 1.12 | n.s |
| OS and hardware | 0.86 | n.s | 0.68 | n.s |
| **Performance** | **2.01** | **1.20e-03** | **2.44** | **1.36e-05** |
| **Security** | **2.27** | **9.71e-09** | **1.97** | **8.01e-06** |
| Tools and extensions | 0.68 | n.s | 1.27 | n.s |

(An OR $>$ 1 indicates that the corresponding topic category is more likely to be posted by end-users on Twitter, and an OR $<$ 1 indicates the opposite; n.s = not significant.)

issues) is observed for Chrome. Detailed results for all topic categories in both subject systems are presented in Table 8. We can clearly see that end-users do not care equally about all issues, and are likely to complain about specific issues. End-users are more concerned about issues related to performance, security and audio/video in both subject systems. In particular, issues related to audio/video are the most sensitive ones to end-users. The development team of the two subject systems may want to pay special attention to code changes that impact modules on audio/video, performance and security, in order to reduce the amount of possible complaints from end-users.

**End-users are less interested in web browser issues related to GUI appearance in both subject systems.** Issues related to GUI appearance (e.g., text appearance, window size, resolution, and browser themes) are more likely reported in non-tweeted bug reports. In Chrome, bug reports containing *coding and debugging* issues are more likely a concern of developers, rather than end-users (as expected). We do not find evidence of this observation in Firefox (OR = 0.70 but $p$-value ¿ 0.05).

As a summary, end-users are more likely to complain about *how well* the web browser works (e.g., performance, security and audio/video), rather than *how it looks* (e.g., GUI appearance). Although our mined topics are for two web browsers, the approach is generalizable to other subject systems. The development team of other systems can apply our approach to mine the topics that their end-users are interested in. The development team can save their effort, if they prioritize development activities by matching the interest of end-users.

*Our results successfully highlight web browser issues that are more sensitive to end-users. For both of our subject systems, end-users are more concerned about how well the web browser works rather than how it looks.*

**RQ3. Does the Development Team Handle a Bug Report Differently if the Problem is Mentioned on Twitter?**

**Motivation** It is unclear if the current practice in bug fixing is impacted by the feedback posted on Twitter by the end-users. It would be interesting to know whether the information provided by end-users on Twitter is currently leveraged in the bug fixing process. Specifically, we examine whether the bugs reported on Twitter receive a different treatment in the bug fixing process from two aspects: the time taken at each stage of the bug fixing process, and the severity or priority of bug reports that is often used for prioritizing the bug reports.

**Approach** To address this question, we define two non-overlapping groups of bug reports based on the presence of associated tweets: 1) non-tweeted group that contains bug reports without associated tweets, and 2) tweeted group that has bug reports with associated tweets. For the second group, we further divide it into two non-overlapping sub-groups: 1) weakly-tweeted group in which bug reports have fewer associated tweets and 2) heavily-tweeted group in which bug reports have more associated tweets. We use the median number of associated tweets as the threshold to obtain the two sub-groups.

Then, we examine whether there is significant difference in the bug fixing process among the different groups in terms of the time aspects and the severity/priority of the bugs.

***1) Time aspects:*** We measure the time aspects of the bug fixing process using three metrics (see Section 4.1): DBR that captures how long it takes for a bug to get the first response from developers; DBA that describes how long it takes to assign the bug to a developer; and DBF that measures how long it takes to fix the bug.

Similarly to RQ2, to study the current treatment of tweets in the bug fixing process, we apply the Fisher's exact test and compute the odds ratio. A contingency table needs to be built for performing the tests. Each contingency table is built using two dimensions (i.e., the number of tweets associated to the bug reports and the speed at which the bug reports are addressed). As we obtain two groups (i.e., non-tweeted and tweeted) or three groups (non-tweeted, weakly-tweeted and heavily-tweeted) from the tweet perspective, we need to obtain several groups for each metric to build the contingency tables. For each metric, we use the median value to separate bug reports into different groups. Then, we can obtain a contingency table based on each metric and the tweet information.

For each metric, we first examine the treatment of bugs based on the presence or absence of tweets associated to the bug reports (i.e., non-tweeted and tweeted). Accordingly, we define and test the following null hypothesis:

$H_0^3 a$ : *There is no difference in the probability of bug reports to be addressed (i.e., responded to, assigned, or fixed) within a certain time duration between bugs with and without associated tweets.*

Second, we examine the treatment of bug reports with different levels of associated tweets (i.e., non-tweeted, weakly tweeted, and highly tweeted) based on each metric. We define the null hypothesis as:

$H_0^3 b$: *There is no difference in the probability of bug reports to be addressed (i.e., responded to, assigned, or fixed) within a certain time duration between bugs with different levels of associated tweets.*

Since we conduct multiple significance tests between the groups of tweeted and non-tweeted bug reports, we correct for the multiple comparisons using the Benjamini-Hochberg correction (Benjamini and Hochberg 1995; Benjamini and Yekutieli 2001).

To further assess the relationship of tweets with the bug fixing intervals, we build a linear regression model to model the bug fixing intervals given a set of predictors (e.g., # of

tweets). The predictors we control for in the regression model are the number of tweets, the severity, the topic, and the affected component of the bug reports. We measure the goodness of fit of the regression model using the coefficient of determination $R^2$. $R^2$ measures how close the fitted regression model is to the actual values of the bug fixing intervals. Then, we report the significance of each predictor on the bug fixing intervals DBR, DBA, and DBF. A small significance ($p$-value) indicates that it is unlikely we will observe a relationship between the predictor (e.g., # of tweets) and the bug fixing interval (e.g., DBF) due to chance.

To verify our findings, we get in touch with developers at Firefox and Chrome. For Firefox, we start a thread at a specialized forum for the contributors in charge of the social media support.[24] Three Firefox contributors were kind to respond to the message posted. For Chrome, we start a thread at the developers forum.[25] Unfortunately, we could not get feedback from the chrome developers. We further investigate the possibility of using tweets to file bug reports. Two weeks after the release of Firefox 53.0.2, we use our approach to collect the bug-reporting tweets. We manually summarize the results from the collected tweets and file 2 bug reports to the issue tracking system Bugzilla.[26,27] We include in the bug reports the links to the users' tweets.

***2) Severity/priority:*** In issue tracking systems, importance tags (i.e., severity and priority) are assigned to bug reports to assist in the bug triaging process. The severity of a bug report describes how damaging a bug is to the system. The priority, on the other hand, defines the order in which a bug should be resolved and deployed. In many cases, severe bugs are also assigned higher priority. In other cases, a bug could possibly have low severity (such as a misspelled title on the home page of a website), but could be assigned high priority because of the visibility of the bug to the end-users. It is also possible for a bug to have high severity, yet be assigned low priority because the bug only occurs in rare occasions. In most cases, only one importance tag is mainly used by software systems in their issue tracking systems.

In Firefox, where severity is the main importance tag, the following severity levels are used to describe the severity of the bug reports: *blocker*, *major*, *critical*, *normal*, *minor* and *trivial*. We identify the severe bugs using the labels *blocker*, *major*, and *critical*. The bugs with normal severity are identified with the labels *normal*, *minor* and *trivial*. In Chrome, the priority tag is used and has the following values: 0, 1, 2, and 3. We classify the bug reports with the priority values 0 and 1 as the most urgent, and the bug reports with values 2 and 3 as the less urgent.

We assess whether the importance level (i.e., severity or priority) of a bug has any association to its number of associated tweets. In each subject system, we classify the bug reports into the highly important bug reports (high severity in Firefox and high priority in Chrome), and the less important bug reports. We use Fisher's exact test to assess whether the presence and absence of associated tweets has an association to the importance level, using the following null hypothesis:

$H_0^3 c$ : *there is no difference in the probability of bug reports to have a certain severity/priority level between bugs with and without associated tweets.*

Similarly, we assess if the different levels of associated tweets are associated to the importance level, by testing the following null hypothesis:

[24] https://support.mozilla.org/en-US/

[25] https://groups.google.com/a/chromium.org/forum/#!forum/chromium-dev

[26] https://bugzilla.mozilla.org/show_bug.cgi?id=1361468

[27] https://bugzilla.mozilla.org/show_bug.cgi?id=1361498

**Table 9** Odds ratio and the corresponding adjusted $p$-value of the Fisher's test on time intervals DBR, DBA and DBF in Firefox and Chrome. (n.s = not significant)

| | Firefox | | | Chrome | | |
|---|---|---|---|---|---|---|
| | DBR | DBA | DBF | DBR | DBA | DBF |
| Non tweeted vs. | 1.08 | 0.94 | 1.19 | 1.13 | 1.09 | 1.10 |
| tweeted | (n.s) | (n.s) | (n.s) | (2.5e-04) | (3.2e-03) | (2.7e-03) |
| Non tweeted vs. | 1.02 | 0.63 | 1.30 | 1.20 | 1.16 | 1.12 |
| highly tweeted | (n.s) | (n.s) | (n.s) | (2.9e-06) | (7.1e-05) | (2.8e-03) |
| Non tweeted vs. | 1.17 | 0.95 | 1.04 | 1.05 | 1.02 | 1.08 |
| weakly tweeted | (n.s) | (n.s) | (n.s) | (n.s) | (n.s) | (n.s) |

$H_0^3 d$ : *there is no difference in the probability of bug reports to have a certain severity/priority level between bugs with different levels of associated tweets.*

Finally, we compute the odds ratio.

**Findings We find no evidence that developers react differently to the bugs that are tweeted in terms of the time aspects of the bug fixing process**. Table 9 shows the odds ratios and adjusted $p$-values resulting from the Fisher's test. We find that, for all time aspects in Firefox, there is no significant difference (i.e., $p$-value>0.05) between tweeted and non-tweeted bugs. Surprisingly, in Chrome, we find that tweeted and highly tweeted bug reports are likely to be addressed (i.e., responded to, assigned, and fixed) in a slower fashion compared to the non-tweeted bug reports. However, the odds ratios are all between 1.09 and 1.20, indicating a low likelihood for tweeted and highly tweeted bug reports to receive a slower response, assignment, and fixing times.

**We find no evidence that the number of associated tweets has a significant relationship with the three bug fixing intervals**. We show in Table 10 the goodness of fit of the three regression models. We observe that the regression model associated with the interval DBF returns the highest goodness of fit (i.e., $R^2 = 0.56$ in Firefox and $R^2 = 0.59$ in Chrome). We further report the significance values of the predictors in Table 10. While the severity of the bug has the highest significance on the bug prediction intervals ($p$-value ¡ 0.05), the number of tweets returns the lowest significance, thus confirming the results obtained from the previous analysis using Fisher's exact test.

**Table 10** Goodness of fit ($R^2$) and signficance results ($p$-value) of the linear regression models (n.s = not significant)

| | Firefox | | | Chrome | | |
|---|---|---|---|---|---|---|
| | DBR $R^2 = 0.56$ | DBA $R^2 = 0.41$ | DBF $R^2 = 0.56$ | DBR $R^2 = 0.39$ | DBA $R^2 = 0.43$ | DBF $R^2 = 0.59$ |
| # of tweets | n.s | n.s | n.s | n.s | n.s | n.s |
| Severity | 1.8e-15 | 3.8e-10 | 8.5e-10 | 4.7e-07 | 9.8e-06 | 6.5e-07 |
| Topic | 1.6e-05 | n.s | 5.3e-06 | 2.7e-03 | 4.1e-04 | 2.4e-03 |
| Component | n.s | n.s | 5.6e-04 | n.s | n.s | n.s |

**Firefox contributors report that bug discovery through Twitter could be challenging.** Unfortunately, we were not able to receive any feedback from the Chrome developers. Considering the lack of evidence showing a relationship between the tweets and the bug fixing intervals (as observed from the previous findings), we get in touch with Firefox contributors who are in charge of the social media support. The contributors report the following difficulties in possibly using Twitter to collect bugs:

> "Quote 1: Tweets may be of interest in so far as if something is reported as a problem with a lot of tweets, it may indicate a severe issue. However, it is a very difficult media to use for communicating information well."

> "Quote 2: Bug discovery using Twitter is not easy. A bug is not as simple as it sounds."

> "Quote 3: As a social media contributor, I've filed bug reports for users that have reported issues via Twitter. The unfortunate fact is that users don't get back to us or continue with the bugs that's been reported on their behalf."

Indeed, tweets are short messages with at most 140 characters and they may be too generic or lack of details to replicate the reported issues. In addition, the high amount of tweets (as shown in Table 1) can be overwhelming, and it might be difficult for developers to manually extract useful information from the tweets. Our approach can help the development team to automatically identify useful feedback from the tweets.

**The bugs reported based on the Twitter feedback are acknowledged by the developers.** The first observed bug is related to the appearance of the URL bar on systems with RTL languages (e.g., Hebrew and Arabic). We receive a response from the Firefox developers 3 days later to inform us that the issue has been fixed and the Twitter user has been notified. The second observed bug after the release of Firefox 53.0.2 is the YouTube music no longer playing after Firefox is sent to the background. We were able to collect complaints from 7 Twitter users. The Firefox developers provided a response on the same day and linked the bug report to an existing bug report. The reported problem has not been fixed yet as of the time of this writing.

**In Firefox, bug reports with associated tweets are more likely to be severe than the non-tweeted bug reports. However, such trend is not observed in Chrome**. Table 11 presents the detailed results of the Fisher's exact test and the odds ratio. In Firefox, bug reports with associated tweets are 1.86 times more likely to be treated as more severe than bug reports without associated tweets. The chance for a bug report to be labeled as severe increases to 2.41 times, if the bug report has a high number of associated tweets (i.e., highly-tweeted). Among the tweeted Firefox bug reports, those that are associated with the more popular tweets (i.e., tweets that are retweeted, marked as favorites, receive a reply from

**Table 11** Fisher's test results regarding the relation between severity/priority level and the level of tweet involvement

| | Firefox (Severity) | | Chrome (Priority) | |
|---|---|---|---|---|
| | Odds ratio | *p*-value | Odds ratio | *p*-value |
| Non tweeted vs. tweeted | 1.86 | 2.02e-03 | 0.98 | n.s. |
| Non tweeted vs. highly tweeted | 2.41 | 1.28e-05 | 0.99 | n.s. |
| Non tweeted vs. weakly tweeted | 1.18 | n.s. | 0.97 | n.s. |

Firefox, and generate active and longer conversations) are likely more severe. The popularity metrics are explained in Section 5.3. Indeed, one of the Firefox contributors suggests in *Quote 1* that a bug could be perceived as severe if reported by many users on Twitter.

> *We do no find evidence of the impact of tweets on the bug fixing process in both subject systems, with the exception of the Firefox tweeted bug reports which are more likely to be severe. Therefore, an automated approach could be used to leverage the feedback from end-users and possibly benefit the bug fixing process*

### RQ4. Can we Use Tweets to Achieve an Early Discovery of Bugs?

**Motivation** The findings of RQ3 indicate that either tweets have no impact on the current practice of the bug fixing process or tweets are not considered useful by the development team. The feedback we receive from the Firefox contributors points toward the second possibility. We are wondering if tweets can still provide some value to the bug fixing process. As tweets provide fast feedback from end-users, the development team may get to know the problems described in tweets earlier than when the problems are currently reported and filed in the issue tracking systems. Even if a 140-character tweet may not sufficiently describe a problem, developers can contact the end-user who posted the tweets for more details. Therefore, we are interested to find out if tweets can lead to an early discovery of bugs.

**Approach** To address this question, we consider two scenarios as described in Fig. 11. The first scenario is the one that we observe in the current practice of the bug fixing process: no relation between the time a bug is reported (i.e., $T_{reported}$) and the time a tweet related to the bug is posted (i.e., $T_{twitter}$). In such a case, a bug can be reported before or after the posting of related tweets. The second scenario describes an ideal case where a bug is filed into the issue tracking system immediately after the problem is posted on Twitter. If the total bug resolution time (i.e., from creation to resolution) remains the same, earlier discovery of bugs might lead to a subset of the bugs to be fixed earlier.

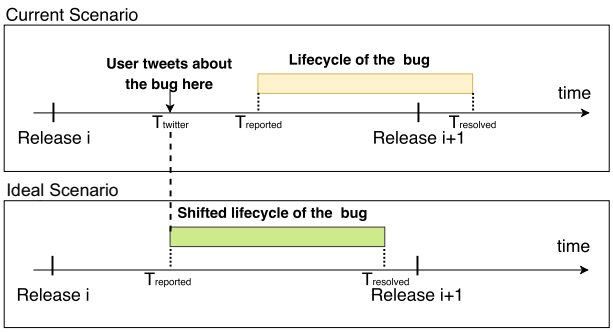



**Fig. 11** An illustration of the real scenario (a bug is not reported at the time the problem appears on Twitter) and the ideal scenario (a bug is reported immediately after the problem is posted on Twitter) for the bug fixing process

The ideal scenario is achievable, as our approach can automatically link tweets and bug reports with a substantial precision. A newly posted tweet may reveal a problem that is unknown to the development team, if no bug reports are retrieved for the tweet. Developers could further contact the end-user who posted the tweets to obtain more details and file a bug report. Even though the Firefox twitter account is quite interactive with the users and Chrome is as well but to a lesser degree, many tweets addressed to both accounts are irrelevant and would require resources to extract, process, and filter. Our collection and filtering approach to essentially collect bug-reporting tweets could help focus the efforts resources toward the relevant feedback from the Twitter users. This would be particularly important after a new release is deployed and users are quick to report problems they face after the browser is auto-updated on their diverse environments. The data collected from Twitter shows that the official Twitter accounts of Firefox and Chrome are only able to reply to 11.9% and 7.4% of the tweets from the users, respectively.

First, we investigate to what extent bugs can be discovered earlier by monitoring tweets. We compute the maximum number of bugs that are reported after their corresponding tweets are posted. Each bug report is associated to zero or more tweets. For each bug report, we use the time of the earliest associated tweet as the earliest time when the bug was reported by an end-user on Twitter. We denote the time as $T_{\text{twitter}}$. The bug reports without associated tweets are excluded from this experiment, as they could not have been discovered through Twitter. We compare the timestamp $T_{\text{twitter}}$ to the timestamp $T_{\text{reported}}$ when a bug is reported, and count the number of bugs that satisfy the condition $T_{\text{twitter}} < T_{\text{reported}}$.

Second, we examine the number of bug fixes that can be delivered in the next release assuming that *a)* a bug is reported immediately after the problem is posted on Twitter, and *b)* the total bug resolution time remains the same. We calculate the updated number of bugs that can be fixed before the next release. For all bug reports with $T_{\text{twitter}} < T_{\text{reported}}$, we intentionally replace the time when a bug is currently reported with the time of the earliest associated tweet (i.e., setting $T_{\text{reported}} = T_{\text{twitter}}$). We keep the duration for each step in the bug fixing process (e.g., bug triaging, fixing, and verification) unchanged. We count the number of bugs that can be "fixed" before the next release, including both the bugs currently fixed before the next release, and the bugs that can be potentially solved before the next release (i.e., when setting $T_{\text{reported}} = T_{\text{twitter}}$).

**Findings With the Lucene mapping that retrieves five bug reports in each query (i.e., $K = 5$), 33.4% and 33.5% of bugs could have been reported earlier in Firefox and Chrome systems, respectively.** Moreover, Firefox bugs could have been reported averagely 8.4 days in advance, and Chrome bugs could have been reported on average 7.6 days earlier. Table 12 shows the percentage of bugs that can be discovered earlier on Twitter and the average time that can be saved to discover bugs.

**Table 12** Summary of RQ3 results

| Subject system | Firefox | Chrome |
|---|---|---|
| # of all bug reports | 14,489 | 15,771 |
| # of mapped bug reports | 6,926 | 8,012 |
| (% of bugs among all bugs) | (47.8%) | (50.8%) |
| # of bugs reported earlier on Twitter | 4,839 | 5,283 |
| (% of bugs among all bugs) | (33.4%) | (33.5%) |
| (% of bugs among all mapped bugs) | (69.9%) | (65.9%) |
| Average improvement (days) | 8.4 | 7.6 |

**A subset of bugs could be fixed earlier, in cases where the development team has the resources to address the extra bugs.** For instance, there are on average 188 bugs fixed before the next release in the 37 Firefox releases; 24 extra bugs (i.e., 212 bugs in total) could possibly be fixed before the next release. In the 34 Chrome releases, the average number of bugs fixed before the next release is 209. An extra 32 bugs (i.e., 241 bugs in total) could possibly be fixed before the next release. The above is an ideal scenario that assumes the availability of resources to fix the extra bugs. In a real life scenario, developers might not have the resources to address all the extra bugs before an upcoming release.

For each subject system, we further perform a paired Mann-Whitney U test to verify the following alternative hypothesis:

$H_a^4$: *The number of bugs that could be fixed before the next release is statistically higher than the current number of fixed bugs.*

The $p$-value is always less than $2.2e$-16 in both subject systems, thus confirming the alternative hypothesis. The number of bugs that could have been fixed before the next release is always greater than the currently fixed bugs in both the Chrome and the Firefox releases. To quantify the magnitude of the observed increase, we calculate the effect size using Cliff's delta (Romano et al. 2006). Cliff's delta is non-parametric, thus it does not make assumptions about the distribution of the data. It is reported to be more robust than Cohen's d (Romano et al. 2006). Cliff's delta reflects the degree of overlap between the two distributions. It ranges from -1 (if all values in the first distribution are larger than the second) to +1 (all values in the first distribution are smaller than the second). When the two distributions are equal, it is equal to 0 (Cliff 1993). Following the guidelines of prior work (Grissom and Kim 2005; Tian et al. 2015; Coelho and Valente 2017), we interpret the effect size $e$ as small for $0.147 < e < 0.33$, medium for $0.33 < e < 0.474$, and large for $e \geq 0.474$. In both browsers, we find a *large* effect size between the two distributions (i.e., the currently fixed number of bugs and the possibly fixed number of bugs).

> *At least a third of the bugs could have been reported to the development team earlier, and a subset of the bugs could possibly be fixed earlier (if resources permit). To achieve this goal, we encourage end-users to promptly report problems on Twitter and the development team to utilize our approach to collect the bug-reporting tweets.*

## 8 Threats to Validity

We discuss the threats to validity of our case study following the common guidelines provided by Yin (2002).

**Threats to conclusion validity** concern the relation between the treatment and the outcome. The conclusion validity threat mainly comes from the inherent errors that come with processing the natural language. Both bug reports and tweets are written by users and are very prone to errors. Tweets are more particularly prone to the use of abbreviations, misspelled words, and non-standard orthography. We attempt to address these concerns by identifying common abbreviations and typos in tweets and correcting the identified misspellings before further processing. Another threat to the validity of the conclusions comes from the use of the bug fixing intervals (e.g., time to fix a bug) to investigate bug reports.

Open source projects are mainly maintained by volunteers who contribute based on their time availability. Therefore, bug fixing intervals are impacted not only by the nature of the bugs, but also by the developers' availability (which we do not consider in this study). To mitigate this threat, we propose to evaluate in future work the impact of end-users' feedback on the bug fixing process using other techniques. Last but not least, since we propose an approach with an average precision value of 79% and unknown recall, we introduce a threat to the validity of the conclusions in RQs 2 to 4. In RQs 2 to 4, we mostly distinguish two groups of bug reports: tweeted and non-tweeted. The main threat of having an unknown recall is mistakenly classifying a bug report as non-tweeted, when in reality it does have tweets associated to it that were missed by the mapping approach. The precision value, on the other hand, indicates that slightly over 20% of the mapped bug reports and tweets are mistakenly mapped. Thus, some of bug reports in the tweeted group might have in reality either *a)* no tweets associated to them, or *b)* a lower number of associated tweets. The latter case is not critical as a bug report with less associated tweets remains a tweeted bug report. However, the former case presents a threat to the validity of the comparisons in RQs 2 and 3. As this is the first attempt to link bug reports and tweets, we wish to improve in our future work the accuracy of the mapping. Besides, we also attempt to verify some of our findings by reaching out to the developers, and by submitting bug reports based on the data we automatically collect from Twitter.

**Threats to internal validity** concern the selection of subject systems and analysis methods. To lower the bias in the manual evaluation in our work, we invite three evaluators (non-authors) to evaluate the mapping between tweets and bug reports, and to manually label the topics generated from the bug reports. It is possible that evaluators could be uncertain about the correctness of a mapping between a tweet and a bug report. Therefore, it is a possible threat to the validity of our results to adopt a binary evaluation (i.e., mapping is either correct or incorrect). We further assess the agreement among the evaluators. We find a substantial agreement, thus adding confidence to the evaluation results. As far as the analysis methods used to pre-process the bug reports (particularly the removal of noise from the bug reports), we are aware of the existence of more sophisticated methods to classify the content of similar content (e.g., development emails) at line level (Bacchelli et al. 2012). We follow a heuristic-based simpler approach in this study that does not require the training of a large dataset at line level, and is tailored to the content of the bug reports. However, we will refine our approach in our future work.

**Threats to external validity** concern the possibility to generalize our results. In this study, we mainly focus on the web browsers Firefox (releases V5.0 to V41.0) and Chrome (releases V15.0 to V48.0), as subject systems. Our findings are specific to these two subject systems, although one of them is a popular example of an open source software, while the other is a proprietary software. Some of the findings might not be directly applicable to other software systems, such as the types of bugs most discussed by users on Twitter. However, our approach can be applied to map tweets to bug reports, and identify the bugs with a large impact on the end-users. In the future, we plan to apply our approach on different types of the subject systems that do not adopt the rapid release cycle.

**Threats to reliability validity** concern the possibility of replicating the study. We attempt to provide all the necessary details to replicate our study.[28] Bugzilla and the

[28] https://github.com/mariamelm2/TwitterPaperRep

Chromium tracker are publicly available to obtain the bug reports. Tweets can also be obtained from Twitter through their search features.

## 9 Implications

**Developers** Development teams strive to provide competitive and good quality software systems. As such, it is important to get the feedback from the end-users in order to prioritize the development efforts, and address the most pressing issues. With the flourishing of the social media platforms, such as Twitter, development teams have an unprecedented access to the feedback from their end-users. Our study provides evidence that *a)* the end-users are likely to vent their frustrations about the web browsers issues on Twitter; *b)* some types of issues (e.g., performance) are more critical to the end-users compared to others (e.g., GUI); *c)* the reported issues can possibly highlight legitimate bugs introduced by the most recent release of a browser; *d)* the reported issues can possibly be reported on Twitter prior to being formally reported on the issue tracking system, thus allowing an early discovery of the browser bugs. In this paper, we propose an approach to collect the bug-reporting tweets from Twitter. With the aforementioned benefits in mind, we encourage the development teams to utilize our approach in order to promptly identify the most pressing issues from the perspective of the end-users after a new version of the browser is released. This could possibly help with the discovery of bugs (i.e., did the developers miss any test cases?), and the prioritization of efforts (i.e., what do most users complain about?).

**End-Users** It is in the best interest of the end-users of any software system to provide feedback (i.e., bug reports and feature requests) to the software providers. Considering the intimidating nature of tools, such as the issues tracking system, social media platforms, such as Twitter (where the end-users are already active) provide a space of unregulated expression and easy access to the software providers. In our study, we observe that the developers are willing to hear from their users, provided there is proper follow-up. Therefore, we encourage the end-users to make use of the power of the social media platforms and promptly report any bugs or feature requests.

## 10 Related Work

It is important to identify software bugs that have high user impact, especially in rapidly evolving software. Leveraging social networks can support to identify the most critical bugs from the perspective of end-users.

### 10.1 Social Networks in Software Engineering

Social networks are web-based platforms where individuals build social relations based on similar interests, activities and backgrounds. Early studies (Ahmadi et al. 2008; Storey et al. 2010) provide insights about the use of social media in SE and clarify the research opportunities that arise from the integration of social media in software development activities. Storey et al. (2010) discuss the different social features that are used in the software engineering practice. For instance, blogs are used by developers to document "how-to" information and discuss the release of new features. Microblogs, such as Twitter, provide

a channel for lightweight coordination and communication. Reinhardt (2009) and (Guzzi et al. 2010) suggest to integrate micro-blogging in IDE to ease communication among developers. Social media services also gradually replace older forms of user support platforms, such as mailing lists. Indeed, Vasilescu et al. (2014) report that mailing lists experts of R have migrated to StackExchange to ask/answer R-related questions. To further benefit from the crowd-sourcing power of social networks, Storey et al. (2010) encourage future research on the role of social media in increasing community and end-user involvement in software engineering. In this paper, we attempt to contribute to this research direction by proposing an automated approach where the feedback from end-users is leveraged to benefit some software engineering activities (i.e., identifying bugs with high user impact).

### 10.2 Micro-Blogging in Software Engineering

Micro-blogging is a communication medium characterized by the exchange of short messages. Micro-blogging has appealed to millions of users due to its immediacy and portability. With the introduction of platforms, such as *Twitter*[29] and *Tumblr*,[30] micro-blogging took off in 2006 and is still going strong. Researchers seeked to understand how software engineering has embraced this medium of communication, and particularly *Twitter*. The software engineering community is a highly interactive population within *Twitter* (Bougie et al. 2011), with different sub-communities discussing specific topics related to software engineering. Based on a survey on developers who are active on Twitter, Singer et al. (2014) find that Twitter can help developers stay up-to-date with the rapidly evolving development technologies, learning, and building professional relationships. The role of Twitter is to disseminate the up-to-date information related to software engineering, as confirmed by Sharma et al. (2015). As opposite to general micro-bloggers, micro-bloggers within the software engineering community form tighter communities where the relationships among the micro-bloggers are not highly reciprocal (Tian and Lo 2014). Beyond understanding the use of micro-blogging in the software engineering community, it is important to leverage the users' posts (e.g., tweets) available on the micro-blogging platforms. However, the noisy content can be an issue when mining micro-blogging data. Therefore, Prasetyo et al. (2012) propose a classifier to identify the micro-blogs related to software engineering based on the texts of the micro-blogs. In this paper, we focus on *Twitter* as it is a prominent micro-blogging medium. We mine and analyze the input of the end-users to the software providers in *Twitter*, rather than the behaviors of the software developers within the platform.

### 10.3 Leveraging User Feedback in Software Engineering

Identifying bugs in software systems is a crucial step in the software quality process. Extensive testing is usually performed to identify defects in the system. The earlier a bug is identified, the lower cost a bug can bring to the software product. However, Aberdour (2007) reports that in the case of an open source software system, defect discovery from black-box testing happens late in the process. Nowadays, social networks provide a convenient way to bridge the communication between developers and end-users. End-users can report bugs or request new features through social networks. It is particularly visible for mobile apps

[29] https://twitter.com/

[30] https://www.tumblr.com/

where users are able to leave ratings and comments on the mobile app stores. A few studies (Villarroel et al. 2016; Chen et al. 2014) have investigated the content from user reviews to improve the development process. Chen et al. (2014) propose AR-miner, a tool designed to identify, group, and prioritize the user reviews. CLAP, a tool proposed by Villarroel et al. (2016), further categorizes the user reviews into bug reports and feature requests. CLAP also clusters the similar reviews together and proposes prioritization for release planning. Keertipati et al. (2016) develop three approaches to construct prioritized lists of features based on the user reviews. Other classifiers (Maalej and Nabil 2015; Panichella et al. 2015) are proposed to identify the relevant and critical user reviews based on the content and sentiment of the reviews, and to identify anger direction in issue reports (Gachechiladze et al. 2017).

In this paper, instead of using the well structured user reviews, we investigate Twitter which records users' feedback in freestyle text. We collect and filter the unstructured feedback from the end-users, and we explore the usefulness of the feedback in reporting bugs as early as possible to developers by capitalizing on the large user base in Twitter.

## 11 Conclusions

Issue tracking systems are mostly used by technical users (e.g., developers) to report bugs in a system. For software systems used mostly by non-technical users (e.g., end-users), social network is a convenient source to collect feedback from users. In this paper, we perform an empirical study to investigate how we can leverage the instant feedback from end-users on Twitter to improve the bug fixing process for rapidly released software (i.e., Firefox and Chrome).

To leverage the user-generated feedback on Twitter, we mine and analyze the tweets that are relevant to the subject systems. Based on our experience of analyzing the tweets, the main challenge of mining tweets is the limited number of characters in the tweets. Even though the tweet itself is too short to contain enough information to file a report, the tweet could point the developers toward the problematic software components. In addition, the developers can reach out easily to the complaining end-users for details about the problems. The advantage of monitoring tweets is to obtain instant end-user feedback and to have access to a large audience of end-users.

First, we propose an approach to map bug-reporting tweets to bug reports. Second, we identify the types of bugs that are most critical to users. We find that end-users are more concerned about how well the systems work (e.g., performance or streaming quality), rather than the appearance of the systems (e.g., resolution or text appearance). Then, we study whether developers respond faster to the bugs that are associated with tweets. We find that the tweeted bugs do not get special attention from developers, suggesting that either there is no close monitoring of bug-reporting tweets or that tweets are not considered as a useful source of information.

Finally, we find that at least 33% of Firefox bugs and Chrome bugs can potentially be discovered earlier based on the prompt tweets from end-users after a new release. This might allow the repair of more bugs before the release of the next browser version. We recommend developers to adopt our approach to extract useful information from tweets, or adapt our approach to collect the readily available and instant feedback that are posted by end-users on other similar social networks.

In the future, we plan to verify our conclusions using other systems for a better generalizability of our findings. We also plan to refine our approach for processing and categorizing the tweets. In particular, we are interested to distinguish tweets that report bugs and tweets that request new features.

## References


Aberdour M (2007) Achieving quality in open-source software. IEEE Soft 24(1):58–64

Ahmadi N, Jazayeri M, Lelli F, Nesic S (2008) A survey of social software engineering. In: ASE Workshops, IEEE, pp 1–12

Arun R, Suresh V, Veni Madhavan CE, Narasimha Murthy MN (2010) On finding the natural number of topics with latent dirichlet allocation: Some observations. In: Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part I, Springer, Berlin, Heidelberg, PAKDD'10, pp 391–402

Bacchelli A, Dal Sasso T, D'Ambros M, Lanza M (2012) Content classification of development emails. In: 2012 34th international conference on software engineering (ICSE), IEEE, pp 375–385

Baldi PF, Lopes CV, Linstead EJ, Bajracharya SK (2008) A theory of aspects as latent topics. SIGPLAN Not 43(10):543–562

Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing . J R Stat Soc Ser B Methodol 57:289–300

Benjamini Y, Yekutieli D (2001) The control of the false discovery rate in multiple testing under dependency. Ann Stat 29:1165–1188

Bies A, Ferguson M, Katz K, MacIntyre R, Tredinnick V, Kim G, Marcinkiewicz MA, Schasberger B (1995) Bracketing guidelines for treebank ii style penn treebank project. University of Pennsylvania 97:100

Bird S (2006) Nltk: The natural language toolkit. In: Proceedings of the COLING/ACL on Interactive Presentation Sessions, Association for Computational Linguistics, COLING-ACL '06, pp 69–72

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3:993–1022

Bougie G, Starke J, Storey MA, German DM (2011) Towards understanding twitter use in software engineering: Preliminary findings, ongoing challenges and future questions. In: Proceedings of the 2Nd International Workshop on Web 2.0 for Software Engineering, ACM, Web2SE '11, pp 31–36

Buckley C, Voorhees EM (2000) Evaluating evaluation measure stability. In: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, SIGIR '00, pp 33–40

Cao J, Xia T, Li J, Zhang Y, Tang S (2009) A density-based method for adaptive lda model selection. Neurocomput 72(7-9):1775–1781

Chen N, Lin J, Hoi SC, Xiao X, Zhang B (2014) Ar-miner: mining informative reviews for developers from mobile app marketplace. In: Proceedings of the 36th International Conference on Software Engineering, ACM, pp 767–778

Cliff N (1993) Dominance statistics: Ordinal analyses to answer ordinal questions. Psychol Bull 114(3):494–509

Coelho J, Valente MT (2017) Why modern open source projects fail. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, New York, NY, USA, ESEC/FSE, vol 2017, pp 186–196

Deveaud R, SanJuan E, Bellot P (2014) Accurate and effective latent concept modeling for ad hoc information retrieval. Document numérique 17:61–84. https://doi.org/10.3166/DN.17.1.61-84

Fleiss JL (1971) Measuring nominal scale agreement among many raters. Psychol Bull 76(5):378

Gachechiladze D, Lanubile F, Novielli N, Serebrenik A (2017) Anger and its direction in collaborative software development. In: Proceedings of the 39th International Conference on Software Engineering. IEEE Press, New Ideas and Emerging Results Track, pp 11–14

Go A, Huang L, Bhayani R (2009) Twitter sentiment analysis. Entropy 17

Griffiths TL, Steyvers M (2004) Finding scientific topics. Proc Natl Acad Sci 101(suppl 1):5228–5235

Grissom RJ, Kim JJ (2005) Effect sizes for research: A broad practical approach. Lawrence Erlbaum Associates Publishers, New Jersey

Guzzi A, Pinzger M, van Deursen A (2010) Combining micro-blogging and ide interactions to support developers in their quests. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, ICSM '10, pp 1–5

Hill F, Reichart R, Korhonen A (2016) Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Computational Linguistics

Keertipati S, Savarimuthu BTR, Licorish SA (2016) Approaches for prioritizing feature improvements extracted from app reviews. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, ACM, p 33

Kouloumpis E, Wilson T, Moore J (2011) Twitter Sentiment Analysis: The Good the Bad and the OMG!, AAAI Press pp 538–541

Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics 33:159–174
Linstead E, Baldi P (2009) Mining the coherence of gnome bug reports with statistical topic models. In: 6th IEEE International Working Conference on Mining Software Repositories. MSR '09., pp 99–102
Liu B, Zhang L (2012) A survey of opinion mining and sentiment analysis. In: Mining text data, Springer, pp 415–463
Maalej W, Nabil H (2015) Bug report, feature request, or simply praise? on automatically classifying app reviews. In: 2015 IEEE 23rd International Requirements Engineering Conference (RE), pp 116–125
MacMahon M, Stankiewicz B, Kuipers B (2006) Walk the talk: Connecting language, knowledge, and action in route instructions. Def 2(6):4
O'Connor B, Balasubramanyan R, Routledge BR, Smith NA (2010) From tweets to polls: Linking text sentiment to public opinion time series. ICWSM 11(122-129):1–2
Pak A, Paroubek P (2010) Twitter as a corpus for sentiment analysis and opinion mining. In: LREC, vol 10, pp 1320–1326
Panichella S, Sorbo AD, Guzman E, Visaggio CA, Canfora G, Gall HC (2015) How can i improve my app? classifying user reviews for software maintenance and evolution. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp 281–290
Picorini C (2015) Lucene-java wiki: Powered by. https://wiki.apache.org/lucene-java/PoweredBy, [Online; accessed 19-December-2016]
Piwowar HA (2011) Who shares? who doesn't? factors associated with openly archiving raw research data. PloS one 6(7):e18,657
Porter MF (1980) An algorithm for suffix stripping. Program 14(3):130–137
Prasetyo PK, Lo D, Achananuparp P, Tian Y, Lim EP (2012) Automatic classification of software related microblogs. In: 28th IEEE International Conference on Software Maintenance (ICSM), pp 596–599
Reinhardt W (2009) Communication is the key - support durable knowledge sharing in software engineering by microblogging. In: Proceedings of the SENSE Workshop, Software Engineering within Social Software Environments, Germany
Romano J, Kromrey J, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In: Annual meeting of the Florida Association of Institutional Research, pp 1–3
Rowe S (2013) Lucene-java wiki: Lucene faq. URL https://wiki.apache.org/lucene-java/PoweredBy, [Online; accessed 19-December-2016]
Schwartz B (2014) A new click through rate study for google organic results
Sharma A, Tian Y, Lo D (2015) What's hot in software engineering twitter space? In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 541–545
Sheskin DJ (2007) Handbook of Parametric and Nonparametric Statistical Procedures, 4th edn. Chapman & Hall/CRC, Boca Raton
Singer L, Figueira Filho F, Storey MA (2014) Software engineering at the speed of light: How developers stay current using twitter. In: Proceedings of the 36th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE, vol 2014, pp 211–221
Socher R, Bauer J, Manning CD, Ng AY (2013) Parsing with compositional vector grammars. In: ACL, vol 1, pp 455–465
Somasundaram K, Murphy GC (2012) Automatic categorization of bug reports using latent dirichlet allocation. In: Proceedings of the 5th India Software Engineering Conference, ISEC '12, pp 125–130
Storey MA, Treude C, van Deursen A, Cheng LT (2010) The impact of social media on software engineering practices and tools. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, ACM, New York, NY, USA, FoSER '10, pp 359–364
Tian Y, Lo D (2014) An exploratory study on software microblogger behaviors. In: 2014 IEEE 4th Workshop on Mining Unstructured Data, pp 1–5
Tian Y, Nagappan M, Lo D, Hassan AE (2015) What are the characteristics of high-rated apps? a case study on free android applications. In: 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, pp 301–310
Vasilescu B, Serebrenik A, Devanbu P, Filkov V (2014) How social Q&A sites are changing knowledge sharing in open source software communities. In: Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing, ACM, CSCW '14, pp 342–354
Villarroel L, Bavota G, Russo B, Oliveto R, Di Penta M (2016) Release planning of mobile apps based on user reviews. In: Proceedings of the 38th International Conference on Software Engineering, ACM, pp 14–24

Weiss C, Premraj R, Zimmermann T, Zeller A (2007) How long will it take to fix this bug? In: Proceedings of the Fourth International Workshop on Mining Software Repositories, IEEE Computer Society, Washington, DC, USA, MSR '07, pp 1

Welch C (2014) Apple pulls ios 8.0.1 after users report major problems with update

Yao X, Van Durme B (2014) Information extraction over structured data: Question answering with freebase. In: ACL (1), Citeseer, pp 956–966

Yin RK (2002) Case Study Research: Design and Methods - Third Edition. 3rd edn. SAGE Publications, Thousand Oaks

Zhang F, Khomh F, Zou Y, Hassan A (2012) An empirical study on factors impacting bug fixing time. In: 2012 19th Working Conference on Reverse Engineering (WCRE), pp 225–234

**Mariam El Mezouar** is a PhD candidate in the School of Computing at Queen's University (Canada). She received her B.S degree in Computer Science and her M.S degree in Software Engineering from Al Akhawayn University (Morocco) in 2011 and 2013, respectively. Her research interests include mining software repositories, collaborative software development, and the social aspects of software engineering.

**Feng Zhang** is currently a postdoctoral research fellow with the Department of Electrical and Computer Engineering at Queen's University in Canada. He obtained his PhD degree in Computer Science from Queen's University in 2016. His research interests include empirical software engineering, mining software repositories, software analytics. His research has been published at several top-tier software engineering venues, such as the IEEE Transactions on Software Engineering (TSE), the International Conference on Software Engineering (ICSE), and the Springer Journal of Empirical Software Engineering (EMSE). More about Feng is available at http://www.feng-zhang.com

**Ying Zou** is the Canada Research Chair in Software Evolution. She is an associate professor in the Department of Electrical and Computer Engineering, and cross-appointed to the School of Computing at Queen's University in Canada. She is a visiting scientist of IBM Centers for Advanced Studies, IBM Canada. Her research interests include software engineering, software reengineering, software reverse engineering, software maintenance, and service-oriented architecture. More about Ying and her work is available online at http://post.queensu.ca/~zouy