# Bug Reports for Desktop Software and Mobile Apps in GitHub: What is the Difference?

Tao Zhang[1,2], Jiachi Chen[2], Xiapu Luo[2], Tao Li[3]
[1]College of Computer Science and Technology, Harbin Engineering University
[2]Department of Computing, The Hong Kong Polytechnic University
[3]School of Software, Nanjing University of Posts & Telecommunications
cstzhang@hrbeu.edu.cn, chenjiachi317@gmail.com, csxluo@comp.polyu.edu.hk,
towerlee@njupt.edu.cn

**Abstract**

With a large number of software products including desktop software and mobile applications (apps), fixing bugs becomes a challenging task. Since bug reports (BRs) can provide more information to fix bugs, researchers leverage BRs to explore how to fix bugs automatically. Previous studies focus on traditional bug tracking systems like Bugzilla rather than GitHub, which is a popular web-based software repository. Moreover, only a few works study BRs for mobile apps whereas most of the previous studies focus on BRs for desktop software. This paper presents a systematic study on the difference in BRs for desktop software and mobile apps hosted in GitHub. First, we collect all BRs from top 100 popular desktop software and mobile apps, and compare their textual features and main elements. Second, we examine the different influence of the BRs' features for desktop software and mobile apps on the fixing time. As a conclusion, we find that commit messages are important resources to help researchers accomplish bug-fixing tasks, such as bug localization for both types of software, especially for desktop software.

**Keywords**

Bug Report, Mobile Applications, Mining Software Repositories, Bug Tracking Systems

## I. Introduction and literature review

As the number of desktop software and mobile apps is rapidly increasing, resolving bugs is a key task. Developers often examine Bug Reports (BRs) in software repositories to get hints for fixing the bugs. The quality of BRs is important. Hooimeijer and Weimer [1] created a descriptive model for the

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

lifetime of a BR. Bettenburg et al. [2] investigated what makes a good BR. Besides, researchers also exploit BRs to develop algorithms for automating various tasks like duplicate detection [3, 4], fixer recommendation [5, 6], feature (*e.g.*, severity/priority prediction) prediction [7, 8], and bug localization [9, 10]. However, these studies usually focus on a few popular projects (*e.g.*, Eclipse) that use traditional bug tracking systems such as Bugzilla. Since GitHub has been widely used to host software, it is desirable to characterize BRs in GitHub and investigate how to perform automated software maintenance tasks for projects in it.

Moreover, most of the previous studies focus on the BRs for desktop software rather than the BRs for mobile apps. To the best of our knowledge, only a few studies [11–13] analyzed the BRs for mobile apps. Syer et al. [11] found that the "best practice" of existing desktop software maintenance cannot be applied to mobile apps due to their different features such as code's size. Bhattacharya et al. [12] conducted an empirical analysis of BRs and bug fixing in Android apps. They investigated the quality of BRs and analyze the bug-fixing process. Zhou et al. [13] conducted a cross-platform analysis of bugs and bug-fixing process of open source projects for Desktop, Android, and IOS. These works mainly studied the BRs for mobile apps in Google Code, which was shut down in January 2016. Recently, more and more mobile apps select GitHub as their management and bug tracking tool. It is worth noting that the analysis results for the BRs in Google Code cannot be used for the BRs in GitHub. The major reason is that the BRs' features (*e.g.*, the number of labels) in GitHub are different from that in Google Code. Moreover, these studies did not investigate the relationship between the main elements (MEs) of BRs and the fixing time of reported bugs. Note that a ME can be a `stack trace`, `code example` or `patch`. It is necessary to analyze the different features of BRs and the relations with the bug fixing activities in these two types of software, because such information can help researchers design algorithms for accomplishing automated software maintenance tasks.

In this paper, we conduct a systematic study on the difference in BRs for desktop software and mobile apps hosted in GitHub. In the first step, we collect BRs in these two types of software and extract the textual features and MEs. Then, we analyze these features and explore the relationship between the features and the bug-fixing time. Moreover, we explain the necessity of adding new data resources such as commit messages. Finally, we point out that commit messages can facilitate automating software maintenance tasks by supplementing more MEs to enrich BRs, especially for desktop software.

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

TABLE I
DATA SCALE

| Date sets overview | | | | |
|---|---|---|---|---|
| Software Type | # Project | # BRs | # Contributors | Average Period/Project |
| Desktop software | 100 | 117,814 | 125,941 | 1033 (days) |
| Mobile apps | 100 | 19,774 | 30,077 | 851 (days) |

| Top-10 popular projects in our data sets | | | | |
|---|---|---|---|---|
| Software Type | Project | # BRs | # Contributors | Period |
| Desktop software | Atom | 8,329 | 329 | 02/02/2012-11/11/2016 |
| | Brackets | 6,221 | 312 | 13/12/2011-09/11/2016 |
| | Chosen | 1,956 | 101 | 18/07/2011-10/11/2016 |
| | Docker | 11,421 | 171 | 20/01/2013-11/11/2016 |
| | Gitlab | 6,668 | 1,127 | 13/10/2011-29/10/2016 |
| | Homebrew | 17,041 | 5,641 | 18/11/2009-01/11/2016 |
| | Lodash | 2,135 | 221 | 28/08/2012-10/11/2016 |
| | Oh-My-Zsh | 3,024 | 1,032 | 21/03/2010-11/11/2016 |
| | Select2 | 3,024 | 360 | 24/03/2013-08/11/2016 |
| | Socket.io | 1,688 | 124 | 01/09/2010-10/11/2016 |
| Mobile apps | ActionBarSherlock | 795 | 53 | 10/03/2011-20/03/2016 |
| | Android-Universal-Image-Loader | 749 | 35 | 08/12/2011-28/09/2016 |
| | Butterknife | 474 | 59 | 27/03/2013-08/11/2016 |
| | Fresco | 1,242 | 63 | 27/03/2015-08/11/2016 |
| | Glide | 1,242 | 43 | 08/08/2013-11/11/2016 |
| | Iosched | 82 | 31 | 10/06/2014-25/08/2016 |
| | MPAndroidChart | 1,918 | 53 | 28/03/2014-10/11/2016 |
| | Picasso | 944 | 73 | 14/03/2013-10/11/2016 |
| | ViewPagerIndicator | 143 | 15 | 04/08/2011-15/06/2016 |
| | Zxing | 525 | 60 | 18/01/2014-09/11/2016 |

## II. DATA COLLECTION

According to the Repositories Ranking (https://github-ranking.com/repositories) in GitHub, we developed a crawler to collect all closed BRs from top-100 popular desktop and mobile apps. Then, according to the apps' description, the second author verifies whether each app really runs in mobile platforms instead of desktop, and filter out the libraries, frameworks and templates related to web because they can be used in both desktop and mobile applications. Finally, the first author checks each decision to ensure a more reliable categorization. Table I lists the information of the collected data. During the periods, the average number of BRs and that of contributors per project in desktop software are much larger than that in mobile apps. Moroever, the average period per project of the former is also larger than that of the latter. We further compare the textual features and MEs of BRs for these two types of software.

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

To help researchers reproduce our work, we publicly share all data sets and analysis results at https://github.com/PolyUCJC317/IEEE-Software.

## III. Textual features in BRs for desktop software and mobile apps

### A. Textual features in BRs

*For each BR, textual features describe what is the bug and how the bug occurs. The main body of BRs are composed of* `title` *and* `description`. `Title` *briefs what is the bug while* `description` *details how the bug occurs.*

The upper part of Fig. 1 shows an example of Atom (desktop software) BR-#6662, and the bottom part shows an example of Fresco (mobile app) BR-#665. Both BRs include `title` and `description`. For instance, BR -#6662 has the title "Cursor position forgotten between tabs" while BR -#665 contains the title "ImagePipelineFactory doesn't use the PlatformBitmapFactory that is set in ImagePipelineConfig". We present their differences in Section III-B.

### B. Comparison of textual features

Textual features help fixers understand why the bug appears and how to fix it. To show the difference, we compare the *length* of BRs for desktop software and mobile apps by counting the number of words in the `description`, because it details the bug. Then, we compute the average number of words (246.4 VS 180.5) per BR in desktop software and mobile apps. Moreover, we calculate the median values (126 VS 95) of BRs' lengths in these two types of software. Note that the average and the median lengths of BRs for desktop software projects are larger than that for mobile apps.

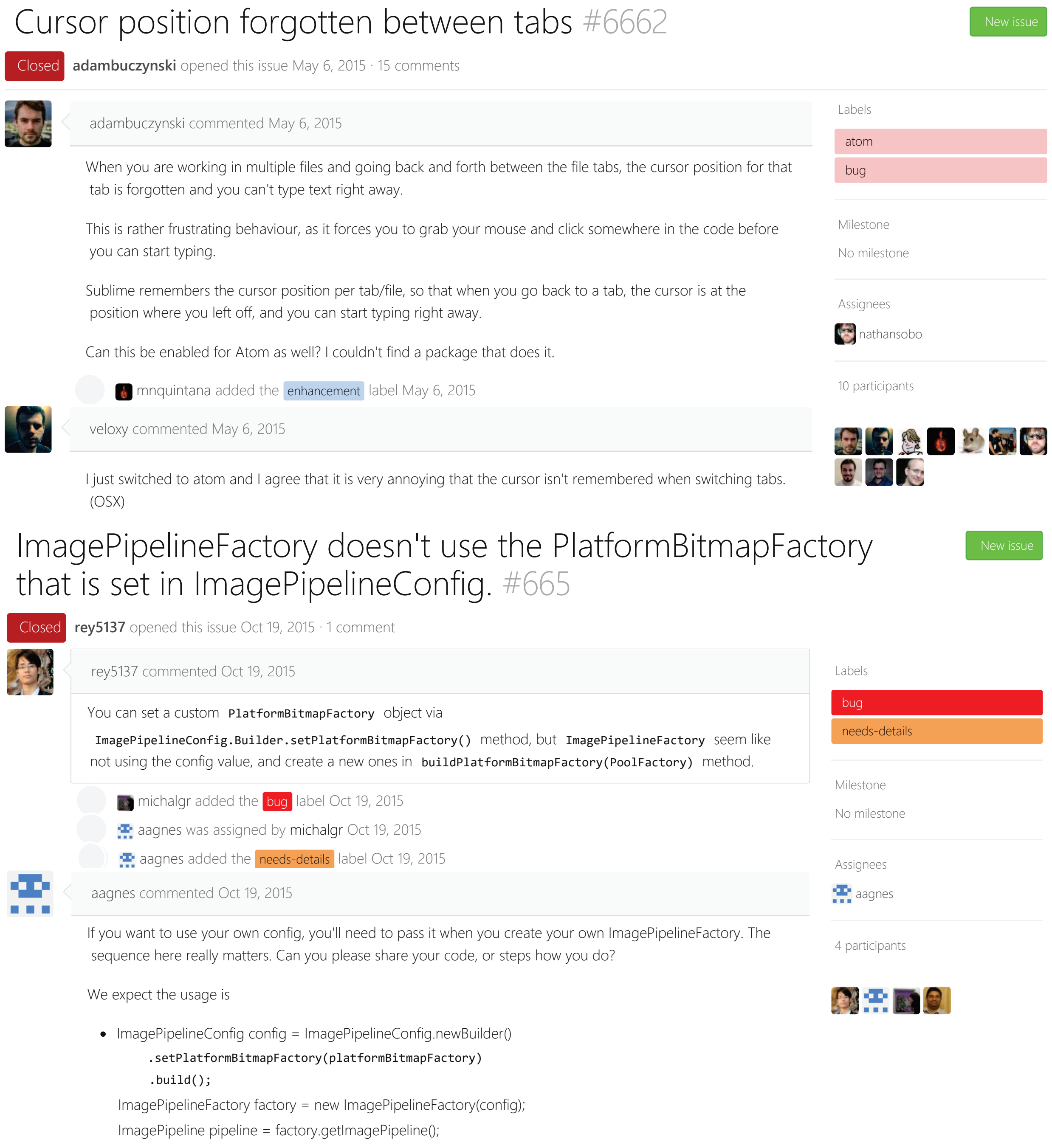
Moreover, we apply hypothesis testing to examine the difference in the lengths of BRs for desktop software and mobile apps. More precisely, we define the null hypothesis: *The lengths of BRs for desktop software show no noteworthy difference against that for mobile apps*. If the p-value is less than the significance level (*i.e.*, 0.05 by default), we can reject the null hypothesis in favor of the alternative hypothesis. Then, we perform two sample t-test [14] for all BRs in two sets, and get a p-value of 2.2e-16. Since it is less than 0.05, we reject this null hypothesis. In other words, the lengths of BRs for desktop software are significantly different from that for mobile apps.

We summarize the BRs' textual features in two different types of software as follows:

| Compared with BRs in desktop software, the average length per BR in mobile apps is shorter. |
|---|

The investigation result poses a question on whether the short BRs for mobile apps affect the bug-fixing process. Before answering this question, we first analyze the BRs' MEs in the next section.

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE



Cursor position forgotten between tabs #6662

New issue

Closed **adambuczynski** opened this issue May 6, 2015 · 15 comments

adambuczynski commented May 6, 2015

When you are working in multiple files and going back and forth between the file tabs, the cursor position for that tab is forgotten and you can't type text right away.

This is rather frustrating behaviour, as it forces you to grab your mouse and click somewhere in the code before you can start typing.

Sublime remembers the cursor position per tab/file, so that when you go back to a tab, the cursor is at the position where you left off, and you can start typing right away.

Can this be enabled for Atom as well? I couldn't find a package that does it.

mnquintana added the enhancement label May 6, 2015

veloxy commented May 6, 2015

I just switched to atom and I agree that it is very annoying that the cursor isn't remembered when switching tabs. (OSX)

Labels: atom, bug

Milestone: No milestone

Assignees: nathansobo

10 participants

ImagePipelineFactory doesn't use the PlatformBitmapFactory that is set in ImagePipelineConfig. #665

New issue

Closed **rey5137** opened this issue Oct 19, 2015 · 1 comment

rey5137 commented Oct 19, 2015

You can set a custom `PlatformBitmapFactory` object via `ImagePipelineConfig.Builder.setPlatformBitmapFactory()` method, but `ImagePipelineFactory` seem like not using the config value, and create a new ones in `buildPlatformBitmapFactory(PoolFactory)` method.

michalgr added the bug label Oct 19, 2015

aagnes was assigned by michalgr Oct 19, 2015

aagnes added the needs-details label Oct 19, 2015

aagnes commented Oct 19, 2015

If you want to use your own config, you'll need to pass it when you create your own ImagePipelineFactory. The sequence here really matters. Can you please share your code, or steps how you do?

We expect the usage is

```
• ImagePipelineConfig config = ImagePipelineConfig.newBuilder()
      .setPlatformBitmapFactory(platformBitmapFactory)
      .build();
  ImagePipelineFactory factory = new ImagePipelineFactory(config);
  ImagePipeline pipeline = factory.getImagePipeline();
```

Labels: bug, needs-details

Milestone: No milestone

Assignees: aagnes

4 participants

Fig. 1. Example BRs for desktop software and mobile apps

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

## IV. MEs in BRs for desktop software and mobile apps

### A. *MEs in BRs*

The quality of a BR is decided by its MEs. Bettenburg et al. [2] summarize that a good BR should include `stack traces`, `code example`, and `patches`.

We explain these features as follows:

- **Stack traces**: A stack trace is a part of the active stack frames at a certain point when a program throws the exception.
- **Code example**: A code example can indicate the reason why the bug appears.
- **Patches**: A patch is a piece of software designed to update a program, it is used to fix (or improve) it.

### B. *Comparison of MEs*

We collect statistics for all BRs, which include the three MEs in our data sets. Specifically, for `stack traces`, we can recognize JAVA stack traces, GDB stack traces, python stack traces, and C++ stack traces. Since the stack traces consist of a start-line and trace lines, we recognize them using regular expressions (*i.e.*, $at + path + : numbers$). For `code example`, we identify C++, python, and JAVA code examples using techniques from island parsing [15]. We can recognize declarations, comments, condition statements, and loops from the programs. `patches` include several start lines indicating the files to be patched and blocks showing the changes to be made. We also used regular expressions (*i.e.*, file types such as `**.**.**.py` and `**/**/**.py`) to recognize them in BRs. We randomly select 200 BRs and double-check the accuracy of the identification method. The results show that the accuracy is 99.5% for `stack traces`, 94.5% for `code example`, and 99.5% for `patches`.

We count the ratio of BRs that contain `stack traces`, `patch`, or `code example` to all BRs in desktop software and mobile apps. The results are shown in Table II. The last column *All ratio* means that the ratio of the BRs that include at least one element to all BRs. Note that the overall ratio of BRs in mobile apps is higher than that of BRs in desktop software. Specifically, even though the ratio of BRs that include `patch` in mobile apps is lower than that in desktop software, the difference is only 1.71%.

We summarize the differences of BRs' MEs in these two types of software as follows:

For MEs, their *all ratio* in BRs for mobile apps is larger than that for desktop apps.

This conclusion raises a question whether the BRs for mobile apps can speed up the bug-fixing process, compared to the BRs for desktop software. We discuss it in the following section.

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

TABLE II
THE RATIO OF MES IN BRS BETWEEN DESKTOP SOFTWARE AND MOBILE APPS

| Platform | Ratio of BRs including stack traces | Ratio of BRs including patch | Ratio of BRs including code example | All ratio |
|---|---|---|---|---|
| Desktop software | 2.85% | 6.12% | 11.64% | 18.47% |
| Mobile apps | 10.13% | 4.41% | 23.40% | 31.75% |

## V. DISCUSSION

### A. *Relationship between BRs' features and fixing time in desktop software and mobile apps*

We investigate the relationship between BRs' features and fixing time. Table III shows the details. The first column shows the length of BRs. The BR-ME ratio describes how many BRs include at least one ME such as `stack traces`, `code example`, and `patches` while the BR-NME (Non-ME) ratio indicates how many BRs do not include any ME.

TABLE III
RELATION BETWEEN BRS' MES AND FIXING TIME

| Software Type | Feature | Length | | | | | |
|---|---|---|---|---|---|---|---|
| | | 0-99 | 100-199 | 200-299 | 300-399 | 400-499 | $\geq$ 500 |
| Desktop software | BR-ME/NME Ratio(%) | 5.75/94.25 | 16.23/83.77 | 24.37/75.63 | 29.96/70.04 | 35.48/64.52 | 54.21/45.79 |
| | Average Fixing Time(days) | 41.94/43.18 | 58.63/62.04 | 65.94/69.51 | 72.26/73.51 | 76.8/80.21 | 88.21/95.80 |
| Mobile apps | BR-ME/NME Ratio(%) | 13.72/86.28 | 37.65/62.35 | 54.15/45.85 | 63.63/36.37 | 70.88/29.12 | 79.18/20.82 |
| | Average Fixing Time(days) | 30.62/31.82 | 33.94/39.99 | 45.37/49.40 | 51.26/52.83 | 51.33/64.46 | 75.95/80.17 |
| Desktop software | Average Number of MEs | 0.06 | 0.17 | 0.26 | 0.33 | 0.40 | 0.65 |
| | Average Size (Words) of MEs | 16.38 | 24.26 | 28.49 | 33.68 | 36.27 | 77.95 |
| | Average Fixing Time(days) | 42.91 | 61.50 | 68.85 | 70.33 | 78.99 | 92.33 |
| Mobile apps | Average Number of MEs | 0.15 | 0.42 | 0.66 | 0.81 | 0.91 | 1.11 |
| | Average Size (Words) of MEs | 29.33 | 46.23 | 71.94 | 85.51 | 93.28 | 174.71 |
| | Average Fixing Time(days) | 30.97 | 37.71 | 48.79 | 50.58 | 55.15 | 79.29 |

We observe that the ME ratio in mobile apps is higher than that in desktop software at all length fields. To verify this observation, we perform a t-test on ME ratios in these two types of software with the null hypothesis: *The ME ratio in mobile apps shows no noteworthy difference against the ME ratio in desktop software*. We get a p-value of 0.00153, which is less than 0.05, and hence we reject this hypothesis.

For both of desktop software and mobile apps, the average fixing time for BRs that include at least one ME is shorter than that for BRs without any ME. This fact indicates that MEs may facilitate shortening bug-fixing time. To verify the relationship between the ME/NME ratio and the fixing time, we conduct a

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

t-test for average fixing time of two sets: BRs containing MEs and BRs without any ME. We define the null hypothesis: *The average fixing time for BRs that include MEs shows no noteworthy difference against the average fixing time for BRs without any ME*. The p-value is 0.01547 and 0.03671 for desktop software and mobile apps, respectively. Since both p-values are less than 0.05, we reject the null hypothesis. In other words, the average fixing time for BRs that include MEs is significantly different from that for BRs without any ME in desktop software and mobile apps.

For BRs that contain MEs, the average fixing time in mobile apps is shorter than that in desktop software. We conduct a t-test with the null hypothesis: *The average fixing time for BRs that include MEs in mobile apps shows no noteworthy difference against that in desktop software*. Since the p-value is 0.0005744, which is less than 0.05, we reject this null hypothesis.

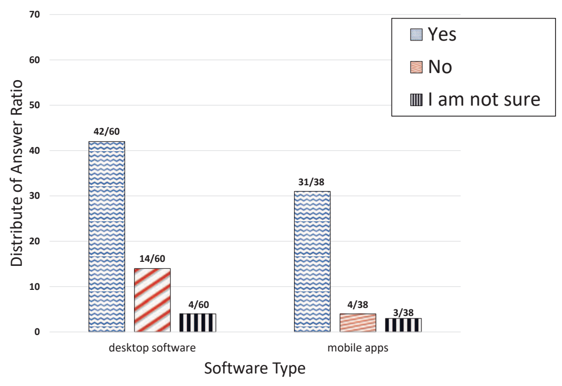
As shown in Table III, with the increase of BRs' lengths, the average fixing time increases. Even though the ME ratio increases for both of desktop software and mobile apps, it is not a major reason of increasing fixing time. Instead, it is due to the small number and size of MEs per BR. For example, when the BRs' length is more than 500 words, the average number of MEs per BR in desktop software is 0.65, and the average size of MEs per BR that includes at least one ME is 77.95. For long BRs, developers have to spend more time on understanding how the bugs are produced and filtering out the noisy data so that the fixing time increases. However, the average fixing time for mobile apps is less than that for desktop software. We execute t-test for the values of $\frac{BR-ME\,Ratio}{Average\,BR\,lenth}$ at each size range for these two types of software with the null hypothesis: *There is no significant difference between desktop software and mobile apps on the relationship of BR-ME ratio and average BR length*. Since the p-value is 0.002074, which is less than 0.05, we reject this hypothesis. The higher ratio of MEs in BRs of mobile apps may be a major reason why their average fixing time is shorter.

To verify whether the MEs can affect the fixing time, we conduct the empirical analysis via a questionnaire survey. We send the online survey https://www.surveymonkey.com/r/CFZN9FN to the top 100 active contributors who posted the largest number of comments to BRs in the desktop and mobile apps of our data sets. Among 98 responses, 60 contributors serve desktop software and 38 contributors work at mobile apps.

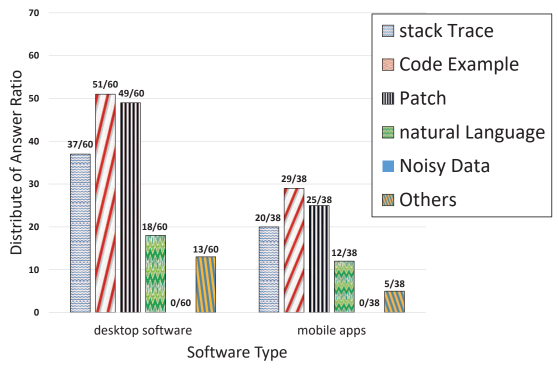
Fig. 2 shows the results of the questionnaire survey. We summarize the answers to **Q1**-**Q3** as follows:

- **Q1**: Most of the contributors in both of desktop software and mobile apps think that the long bug reports delay the fixing time.
- **Q2**: Most of the contributors in both of desktop software and mobile apps think that the MEs such as stack trace, code example, and patch shorten the fixing time.

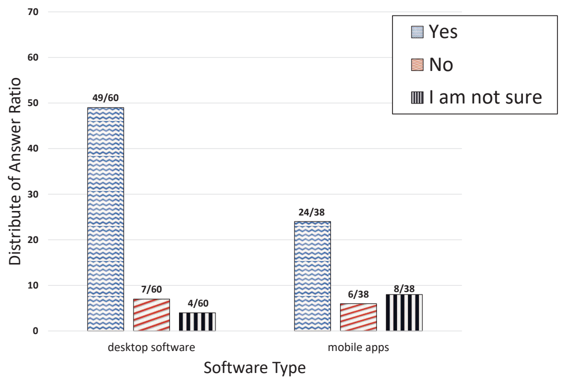
Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE



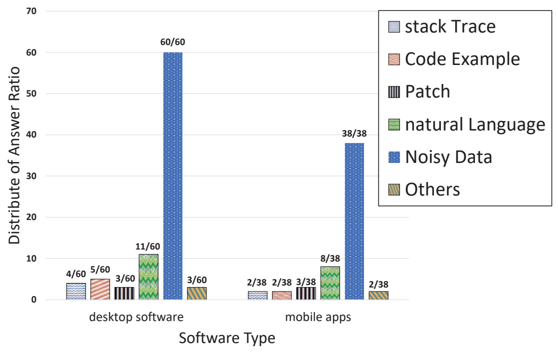
(a) Q1: Do you think whether the bug reports with long-length delay the fixing time?



(b) Q2: Which element(s) can help you shorten the fixing time?



(c) Q3: Which element(s) can delay the fixing process?



(d) Q4: Do you think whether historical commit messages related to reported bugs can help you shorten the fixing time?

Fig. 2. Results of questionnaire survey

- **Q3**: Most of the contributors in both of desktop software and mobile apps think that noisy data harms the fixing process.

*Others* in Fig. 2(b)-(c) refers to other factors. For desktop software, 5 contributors think that BRs' quality, writing style, developers' experience, root cause, and bugs' recurrence can shorten the fixing time whereas 8 contributors give the answer "None of the above". In addition, 3 contributors summarize that human factor delays the fixing process. For mobile apps, 1 contributor thinks that root cause can shorten the fixing time while 4 contributors give the answer "None of the above". Moreover, 2 contributors think that the correctness of BRs' descriptions and human factor delay the fixing process.

The investigation results demonstrate that lack of MEs is the major reason why the average fixing time in desktop software is longer than that in mobile apps.

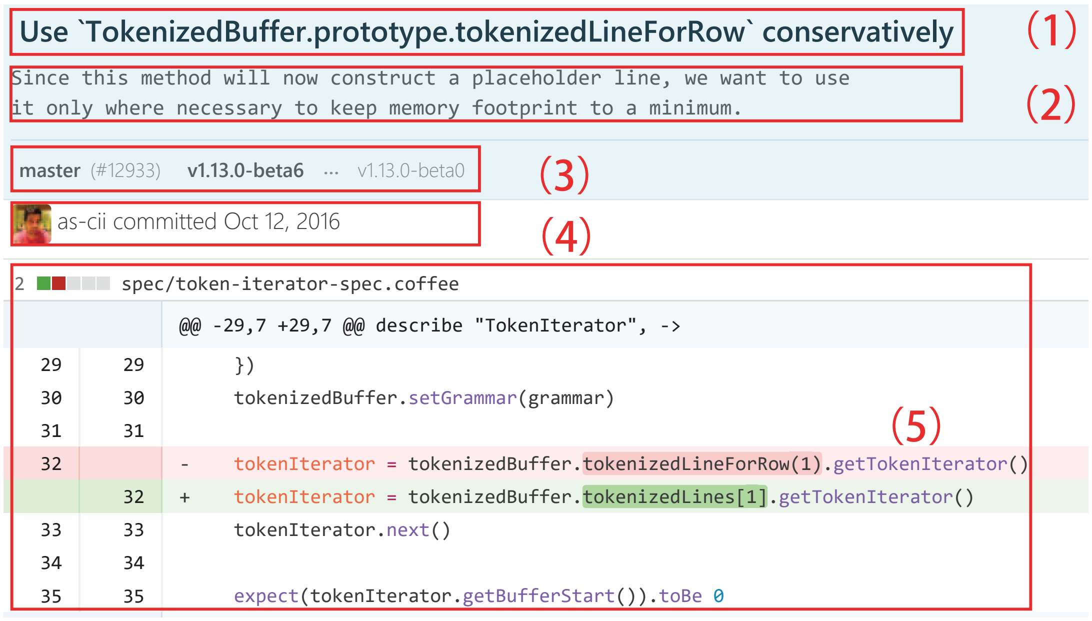
Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

### *B. New resource: commit message*

The analysis of the relationship between BRs' features and bug fixing motivates us to suggest adding additional data resources to enrich BRs, especially for desktop software, for resolving bugs. The data resources should have two important characteristics:

- **Enrichment**: the new data resources should contain rich descriptions of bugs, *i.e.*, high ratio of MEs.
- **Relevancy**: the contents of new data resources should be related to the reported bugs in BRs and are not noisy data.

Commit messages record what information has been changed in source code. Fig. 3 shows an example of the commit message that includes five parts: 1) Title, 2) Description, 3) Feature, 4) Committer, and 5) Changed code.



Use `TokenizedBuffer.prototype.tokenizedLineForRow` conservatively

```
Since this method will now construct a placeholder line, we want to use
it only where necessary to keep memory footprint to a minimum.
```

master (#12933) v1.13.0-beta6 ... v1.13.0-beta0

as-cii committed Oct 12, 2016

```
2 spec/token-iterator-spec.coffee
@@ -29,7 +29,7 @@ describe "TokenIterator", ->
      })
      tokenizedBuffer.setGrammar(grammar)

-     tokenIterator = tokenizedBuffer.tokenizedLineForRow(1).getTokenIterator()
+     tokenIterator = tokenizedBuffer.tokenizedLines[1].getTokenIterator()
      tokenIterator.next()

      expect(tokenIterator.getBufferStart()).toBe 0
```

Fig. 3. An example of commit message

By analyzing the textual contents shown in Fig. 3, we note that the commit messages meet the characteristics of the required new data resources. For example, this case contains a `code example-TokenizedBuffer.prototype.tokenizedLineForRow`, which is related to the bug reported in # 12933. Moreover, the code example enriches bug report # 12933 and can help to fix the bug.

For the answer of **Q4** as shown in Fig. 2, most of the contributors in desktop software think that historical commit messages are useful to shorten the fixing time, and the ratio is higher than that in

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

mobile apps. The result is in consistence with the statistical analysis on the relation between BRs' main features and fixing time shown in Table III. Compared with desktop software, the ratio of MEs in BRs of mobile apps is higher, and the average fixing time is shorter. Thus, the contents of BRs in mobile apps may be enough for developers to resolve the reported bugs.

As a conclusion, historical commit messages can enrich related BRs especially in desktop software because they can provide more detailed information (*e.g.*, MEs) about the reported bugs. Using them, researchers can develop algorithms to automate software maintenance tasks such as bug localization, fixer recommendation, and severity prediction for GitHub software, especially for desktop software.

## VI. Conclusion

We conduct a systematic investigation on the difference between the BRs for desktop software and that for mobile apps in GitHub. Besides discussing the influence of the BRs' features, we investigate the relationship between BRs' features and the time of bug fixing. We found that the average fixing time for desktop software is longer than that for mobile apps due to the lack of high-ratio MEs contained in BRs of desktop software. The results indicate that it is necessary to add new data sources to enrich BRs, especially for desktop software. By doing so, developers could improve the effectiveness of automated software maintenance tasks, such as fixer recommendation and bug localization. We find that commit messages can facilitate these tasks for GitHub projects, especially for desktop software. For instance, researchers can utilize the useful information (*e.g.*, code example) extracted from commit messages to improve the accuracy of bug localization using BRs. In future work, we will use commit messages to develop effective algorithms for automating these software maintenance tasks and compare it with traditional approaches for GitHub projects.

## VII. Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 61602258 and Grant 61202396, in part by the Fundamental Research Funds for the Central Universities under Grant HEUCFJ170604, in part by the Hong Kong GRF (No. PolyU 5389/13E, 152279/16E), and in part by the Shenzhen City Science and Technology R&D Fund under Grant JCYJ20150630115257892.

## References

[1] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE'07*, 2007, pp. 34–43.

[2] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *FSE'08*, 2008, pp. 308–318.

[3] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *ICSE'08*, 2008, pp. 461–470.

[4] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *MSR'13*, 2013, pp. 183–192.

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *ICSE'06*, 2006, pp. 361–370.

[6] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "Dretom: Developer recommendation based on topic models for bug resolution," in *PROMISE '12*, 2012, pp. 19–28.

[7] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *MSR'10*, 2010, pp. 1–10.

[8] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *WCRE'12*, 2012, pp. 215–224.

[9] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports," in *ICSE'12*, 2012, pp. 14–24.

[10] R. Saha, M. Lease, S. Khurshid, and D. Perry, "Improving bug localization using structured information retrieval," in *ASE'13*, 2013, pp. 345–355.

[11] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams, "Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source android apps." in *CASCON'13*, 2013, pp. 283–297.

[12] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source android apps," in *CSMR'13*, 2013, pp. 133–143.

[13] B. Zhou, I. Neamtiu, and R. Gupta, "A cross-platform analysis of bugs and bug-fixing in open source projects: desktop vs. android vs. ios," in *EASE'15*, 2015, p. 7.

[14] C. A. Boneau, "The effects of violations of assumptions underlying the t test." *Psychological bulletin*, vol. 57, no. 1, p. 49, 1960.

[15] L. Moonen, "Generating robust parsers using island grammars," in *WCRE'01*, 2001, pp. 13–22.

## BIOGRAPHY

**Tao Zhang** is an associate professor in the College of Computer Science and Technology at Harbin Engineering University. He is also an research associate in the Department of Computing at Hong Kong Polytechnic University. His research interest includes mining software repositories and empirical software engineering. Zhang received a PhD in computer science from University of Seoul, South Korea. Contact him at cstzhang@hrbeu.edu.cn, sites.google.com/site/toddtzhang/home.

**Jiachi Chen** is a master student in the Department of Computing at Hong Kong Polytechnic University. His research interest includes program analysis and network security. Chen received his bach-

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE

elor's in the Institute of Service Engineering at Hangzhou Normal University. Contact him at chenjiachi317@gmail.com.

**Xiapu Luo** is an assistant professor in the Department of Computing at Hong Kong Polytechnic University. He is also affiliated with the Shenzhen Research Institute of the Hong Kong Polytechnic University. His research focuses on Android security, network security and privacy, and Internet measurement. Luo received his PhD in Computer Science from the same university and then spent two years at the Georgia Institute of Technology as a postdoctoral fellow. Contact him at csxluo@comp.polyu.edu.hk. He is a corresponding author of this paper.

**Tao Li** is a professor in the school of Computer Science and Technology at Nanjing University of Posts and Telecommunications. He is also a professor in the School of Computing and Information Sciences at Florida International University. His research interests include data mining, computing system management, information retrieval, and machine learning. He received his PhD in Computer Science from University of Rochester, Rochester, NY. He received the US National Science Foundation (NSF) CAREER Award and multiple IBM Faculty Research Awards. Contact him at towerlee@njupt.edu.cn.

Digital Object Identifier 10.1109/MS.2017.377142400 0740-7459/$26.00 2017 IEEE