What are the Features of Good Discussions for Shortening Bug Fixing Time?*

Yuki NOYORI††, Nonmember, Hironori WASHIZAKI†, Yoshiaki FUKAZAWA†, Hideyuki KANUKA†††, Keishi OOSHIMA††, Members, Shuhei NOJIRI†††, and Ryosuke TSUCHIYA†††, Nonmembers

SUMMARY Resource limitations require that bugs be resolved efficiently. The bug modification process uses bug reports, which are generated from service user reports. Developers read these reports and fix bugs. Developers discuss bugs by posting comments directly in bug reports. Although several studies have investigated the initial report in bug reports, few have researched the comments. Our research focuses on bug reports. Currently, everyone is free to comment, but the bug fixing time may be affected by how to comment. Herein we investigate the topic of comments in bug reports. Mixed topics do not affect the bug fixing time. However, the bug fixing time tends to be shorter when the discussion length of the phenomenon is short.

key words: bug report, pattern mining, text mining, discussion

1. Introduction

Bug reports allow users to indicate defects in software. Developers then modify software by referring to these reports. Because many bug repositories used in open source software support comments such as questions or opinions about the revisions when making modifications, developers often discuss bugs. Some bugs are quickly modified, while others take a long time to address. Software developers should solve defects efficiently. However, software is often released with defects since software developers do not have sufficient time to modify bugs[1].

Various studies have investigated bug reports. Many have focused on solving defects efficiently. Examples include predicting bug report severity, detection of duplicate bug reports, bug report summarization, bug localization, etc. [2]–[6], [24], [25]. The topic most relevant to our research is the quality of bug reports. Some works have measured the quality of a bug report based on whether readability or code examples are included[1], [7]. These studies measured the quality for the first reported document, but did not consider the comments. Other studies have exam-

Copyright © 2021 The Institute of Electronics, Information and Communication Engineers
sumptions, we aimed to answer the following five research questions (RQs):

- **RQ1**: Is there a relationship between the number of sentences in each topic in the bug report comments and the bug fixing time?
- **RQ2**: Are topics mixed in the meaning paragraph of bug report comments?
- **RQ3**: Is there a relationship between mixed topics in the meaning paragraph and the bug fixing time?
- **RQ4**: Do bug report comments have typical patterns?
- **RQ5**: Is there a way to advance the discussion to realize a short bug fixing time?

The rest of this paper is structured as follows. Section 2 describes our motivating example. Section 3 overviews our method, while Sect. 4 presents our experiments and results. Section 5 considers threats to validity. Section 6 lists potential applications. Section 7 presents related works. Finally, Sect. 8 concludes the paper and provides future work.

### 2. Motivating Example

Software users report bugs that developers use to make modifications. Some bugs are modified immediately, while others are not. Figure 3 shows the beginning of an example of an Eclipse\(^1\) bug report, which was modified soon after being reported. In fact, this bug report was closed in two days. It contained descriptions of highly reproducible phenomena as well as the cause of the defect. A patch was immediately created and discussed in the comments.

---

\(^1\)Eclipse is a trademark of the Eclipse Foundation.
Fig. 3 Example of a bug report (beginning part of bug 276131 from the Eclipse Bug Repository).

Fig. 4 Example of a bug report (beginning of bug 156905 from the GIMP Bug Repository).

Figure 4 shows the beginning of an example of a GNU Image Manipulation Program (GIMP) bug report that took a long time to fix. This bug report was closed after 1407 days. Although the phenomenon and cause were reported at the beginning, the description prior to proposing a solution was unclear.

These examples demonstrate that the bug fixing time greatly depends on the bug report. Ideally, bugs should be solved quickly because developers due to resource limitations. Although previous studies have investigated the initial description for efficient bug fixing, this study focuses on comments. Because discussions are included in the process of bug resolution, we use the above RQs to investigate the relationship between bug fixing time and bug report comments to help developers.

We speculate that the bug fixing time is related to the topic of each sentence in the bug report. Previous studies revealed that the readability is related to the quality of bug reports [7]. Hence, we investigate whether topics are mixed in a meaning paragraph. Intuitively, if topics are mixed, developers become confused, which will increase the bug fixing time. Additionally, we checked whether the comments have patterns and if these patterns affect the bug fixing time.

3. Study Design

To support bug fixes, we analyzed the comments in bug reports from various perspectives. Spearman’s rank correlation analysis was used to consider the bug report comment topic. We also investigated topic mixing in the meaning paragraph using natural language processing techniques to evaluate the confusion of developers when topics are mixed. If a developer becomes confused, the bug fixing time may be negatively impacted. We also investigated how to advance the discussion to efficiently solve bugs. We analyzed discussions using pattern mining to identify if there are ways to develop bug reports with short bug fixing times.

3.1 Dataset

We used bug reports freely available. We acquired bug reports from four Open Source Software (OSS): Eclipse Platform [20], Gnome† [21], Mozilla†† [22], and KDE††† with random sampling. As a result of the calculations based on the interval estimation of the population proportion with 95% confidence level, the appropriate sample size was 96 bug reports. To evaluate whether a project has unique features, we collected 96 bug reports for the Mozilla projects and 24 bug reports for each of the other three projects. We collected a total of 168 bug reports. We conducted two experiments: one using only bug reports for the Mozilla projects and one that spans across the four OSS projects. In each experiment, the sample size was 96 bug reports. The shortest bug fixing time was 1 day, whereas the longest was 5447 days. The average value was 125 days, and the median value was 15 days. Prior to the analysis, we labeled each sentence by its content. These categories are the topic of each comment in the bug report. Similar to our previous work [8], we used five content categories:

†GNOME is a trademark of the GNOME Foundation.
††Mozilla is a trademark of the Mozilla Foundation.
†††KDE is a trademark of KDE.e.V.
††††Standard Widget Toolkit
NOYORI et al.: WHAT ARE THE FEATURES OF GOOD DISCUSSIONS FOR SHORTENING BUG FIXING TIME?

Table 1  Example of a sentence corresponding to each category in a bug report

| Sentence                                                                 | Category |
|--------------------------------------------------------------------------|----------|
| When I try to export reports that contain graphs, gnuCash just creates an empty html file. | Phenomenon (P) |
| The problem here is that nsSearchSuggestions.js is passing the wrong previousResult to form the history. | Cause (C) |
| Perhaps the page should take CSS from intl.css, where localisers can override some of the classes or add a class to a set direction. | Solution (S) |
| Would setting the value to an empty-string("") or something similar work? | Discussion (D) |
| Sounds fine to me. I guess…                                             | Other (O) |

• Phenomenon (P): P represents the behavior of the encountered bug. Most phenomena are written at the beginning of a report or after someone else has confirmed the bug.

• Cause (C): C is the sentence describing why the bug occurred. In some cases, the cause is specified at the beginning. In others, it is the result of an investigation. This is the least common bug report category in this analysis.

• Discussion (D): D is a conversation on how to fix a bug. It includes questions and answers.

• Solution (S): S devises a method to fix a bug such as a patch. Although some examples appear in the first half of a bug report, most are in the second half.

• Other (O): O denotes remarks unrelated to bugs such as greetings and gratitude.

We labeled each sentence and analyzed each bug report. Table 1 shows an example of bug report sentences corresponding to each category, where “Category” in the left column denotes manual labeling.

3.2 RQ1. Relationship between Bug Fixing Time and Topic

First, we examined the relationship between bug fixing time and topic. We assumed that if a particular topic has a long sentence, that developers are confused and it takes a long time to fix the bug. Specifically, we investigated whether the number of sentences in each topic and the bug fixing time are related using correlation analysis and statistical tests.

3.3 RQ2. Mixed Topics in the Meaning Paragraph

While reading a bug report, we noticed some topics in the meaning paragraph are mixed. An intuitive understanding is challenging when topics are mixed. Thus, we examined the presence of mixed topics in bug reports. Each report was divided into meaning paragraphs, which are a collection of sentences within a category. A mixed topic is defined as a meaning paragraph containing three or more categories.

A natural language processing technology called Text Segmentation was used to divide the sentences into meaning paragraphs [9]. Herein Texttiling was used for text segmentation [10]. Texttiling is a technique to compare cosine similarities by moving two chunks composed of several adjacent words. Hence, it can determine whether topics are mixed in each bug report.

3.4 RQ3. Relationship between Bug Fixing Time and Topic Mixing in the Meaning Paragraph

After confirming the existence of bug reports with mixed topics, we investigated whether such bug reports take a long time to fix. We assumed that developers who read bug reports with mixed topics are confused and cannot efficiently fix bugs. Here, topics mean the five categories above. Specifically, we compared the bug fixing time for bug reports with and without topic mixing. Similar to RQ1, we employed correlation analysis and statistical tests.

3.5 RQ4. Pattern for Bug Report Comments

Bug reports generally have a flow where a phenomenon is initially reported followed by a discussion and finally a solution. Hence, typical patterns may exist in the discussion of bug reports that affect the bug fixing time. We focused on the progress of the discussion via pattern analysis. We conducted sequential pattern mining, which is a technique to extract patterns and rules for time series data [11]. Unlike association analysis, time series can be considered [12].

3.6 RQ5. Relationship between Bug Fixing Time and Patterns for Bug Report Comments

If typical patterns exist, are they related to the bug fixing time? To extract the ideal way to discuss bug fixing, we divided the bug reports into those with a short bug fixing time and those with a long bug fixing time. We conducted sequential pattern mining for each collection of bug reports. Then typical patterns present in only those with short or long bug fixing times were extracted, assuming that the extracted patterns are related to the bug fixing time.

4. Findings

Figure 2 overviews the process to answer the RQs. We conducted a U-test to determine if the difference between the two groups is significant. We calculated the p-value from the results of the U-test. The 5% level was judged to be a significant difference between a long bug report and a short bug report.
4.1 RQ1. Relationship between Bug Fixing Time and Topic

As the first step, we analyzed the number of topic sentences by category and the bug fixing time. We counted the number of phenomena, cause, discussion, solution, and other sentences in each bug report. We conducted correlation analysis of the number of topic sentences and the bug fixing time. For example, Fig. 2 shows that the numbers for phenomenon, cause, discussion, solution, and other in the first bug report are 9, 4, 3, 9, and 4, respectively. The fixing time is 20 days. We counted all bug reports in our dataset.

Table 2 shows the number of sentences in each category of the bug reports for the Mozilla projects. Table 3 shows the number of sentences in each category of the four projects. The category of Cause (C) has the fewest sentences, whereas other (O) has the most. Table 4 and Table 5 show the results of correlation analysis, including the factors and correlation coefficients. Most values are low. Among the relationships between bug fixing time and topic, discussion (D) and other (O) in the Mozilla project have the largest absolute value of correlation coefficients (0.563). Looking at the relationship between bug fixing time and the number of sentences in each category, the correlation coefficient is generally low. The bug fixing time is not correlated to a specific topic.

We also calculated the results of the U-tests. Figure 5 shows the distribution of the bug fixing time for all datasets. The x-axis is the bug report, and the y-axis is the bug fixing time. Since the bug fixing time increases remarkably after 100 days, we compared whether or not the time exceeded 100 days. When divided by 100 days, 131 bug reports have a short bug fixing time, and 37 have with a long bug fixing time. Table 6 shows the results. The lowest p-value (0.001) is for Discussion (D). The p-value is low for solution (S), discussion (D), and other (O). However, the bug fixing time is not correlated to any category. This indicates that even if a user writes a lot of phenomena, the bug fixing time will not be shortened. Hence, there is no need to intentionally write many phenomena.

Results of RQ1:

“The number of sentences in each topic in the bug report comments and the bug fixing time are unrelated”.

4.2 RQ2. Mixed Topics in the Meaning Paragraph

We conducted text segmentation to assess whether topics are mixed in the meaning paragraph. Figure 6 shows an example where text segmentation for mixed topics. Although
Fig. 6 Example of text segmentation result (a part of bug 250125 from the Eclipse bug repository)

paragraph 1 consists of one topic, but paragraph 2 mixes three topics: phenomenon, cause, and discussion. Text segmentation showed that 56 bug reports for the Mozilla projects have mixed topics, while 58 bug reports for the four projects have mixed topics. This is the answer for RQ2. Additionally, some bug reports contain only two paragraphs. A bug report with mixed topics may affect the bug fixing time.

Results of RQ2:
“Topic mixing in the meaning paragraph of the comments occurs in some bug reports.”

4.3 RQ3. Relationship between Bug Fixing Time and Mixed Topics in the Meaning Paragraph

We performed correlation analysis on the presence of mixed topics and the time to fix bugs (Table 7). The correlation coefficient of the Mozilla projects is 0.049 and the correlation coefficient of the four projects is 0.018. We also conducted statistical tests. We performed a U-test on the fixing time by dividing it into two: those with and without mixed topics. The p-value of the Mozilla projects is 0.672, while that of the four projects is 0.878. These results are confirmed that mixing topics are unrelated to bug fixing time.

If we attached a category with a different granularity such as non-functional requirements, we may identify a relationship. Mixed topics do not create confusion for developers with content categories. However, other factors may cause confusion. For example, if the description of phenomena or its cause is incorrect, intuitively it is likely that the confusion of developers will affect the bug fixing time. Since several studies examined the quality of the first bug report, in the future, we would like to analyze bug reports using our experimental results and the results of other research on the quality of bug reports.

Results of RQ3:
“There is no relationship between mixed topics and the bug fixing time.”

4.4 RQ4. Pattern for Bug Report Comment

Intuitively, a bug report is solved by reporting a phenomenon, discussing it among developers, and providing a solution. Hence, we used sequential pattern mining to determine whether bug reports have typical patterns (Table 8). It should be noted that the table only includes high support values, which are the frequency of occurrence of a given rule. Here, these are phenomenon P, cause C, discussion D, solution S, and other O. Many rules have high support value and include O. A typical bug report pattern is where S comes after P. Using such sequential pattern mining, we revealed typical patterns in bug reports.

Results of RQ4:
“There are typical patterns in the discussion of bug reports.”

4.5 RQ5. Relationship between Bug Fixing Time and Patterns for Bug Report Comments

We verified whether specific patterns are related to the bug
fixing time by comparing the results of sequential pattern mining only for bug reports with a short bug fixing time to those with a long bug fixing time. We extracted only support values of 0.5 or more. Rules appearing in both the long and short bug fixing times were excluded. Table 9 shows the extracted rules from the bug reports for the Mozilla projects with a long bug fixing time. Table 10 shows the extracted rules from the bug reports for the four projects with a short bug fixing time. Table 11 shows the extracted rules from the bug reports for the four projects with a long bug fixing time. Table 12 shows the extracted rules from the bug reports for the four projects with a short bug fixing time. Table 8 shows that the $P \rightarrow D$ pattern exists in most bug reports. Tables 9 and 10 highlight the nature of the Mozilla bug report discussion. For the Mozilla projects with a long fixing time, the discussion occurs after the solution (Table 9). On the other hand, the Mozilla projects with a short bug fixing time have a pattern of $P, C \rightarrow S$ (Table 10). Hence, the solution topic begins after the cause of the bug is described.

We extracted the universal characteristics of bug reports from Tables 11 and 12. Bug reports with a long fixing time tend to discuss the phenomenon and have a pattern of $D, S \rightarrow S$ (Table 11). Bug reports with a short bug fixing time tend to have a topic of cause (Table 12). Consequently, the discussion patterns depend on the bug report project. Furthermore, in a bug report with a short bug fixing time, the topic of the cause is included in the pattern. It is assumed that the cause is smoothly determined and the discussion of the phenomenon is short. Therefore, we further compared the length of the discussion of the phenomenon.

Bug reports with a short bug fixing time may have a short discussion of the phenomenon. We verified this assertion using the bug reports by reading the categories and investigating the number of conversations until the solution was started. We compared the conversation length until the solution was started between bug reports with a long bug fixing time to those with a short bug fixing time. Similar to the other experiments, the reports were divided by 100 days. Figure 7 shows the results of the bug reports for the Mozilla projects, while Fig. 8 shows the results of the bug reports for the four projects.
Table 12  Extracted rules from the bug reports for the four projects with a short bug fixing time.

| Rule       | Support | Confidence | Lift  |
|------------|---------|------------|-------|
| P, O → S  | 0.964   | 0.964      | 0.964 |
| P, P → P  | 0.909   | 0.909      | 0.909 |
| S, D → D  | 0.545   | 0.857      | 0.962 |
| C → O     | 0.527   | 1          | 1     |
| C, O → O  | 0.527   | 1          | 1     |
| P, C → O  | 0.527   | 1          | 1     |
| C → S     | 0.509   | 0.966      | 0.966 |
| P, C → S  | 0.509   | 0.966      | 0.966 |

Table 13  U-test result of the conversation length until the solution starts

|                    | p-value   |
|--------------------|-----------|
| Mozilla Projects   | 0.000037  |
| Four Projects      | 0.000153  |

Fig. 7  Conversation length until the solution starts for the Mozilla projects.

Fig. 8  Conversation length until the solution starts for the four projects.

Fig. 9  Number of words in the first report for the Mozilla projects.

Fig. 10 Number of words in the first report for the four projects.

Bug reports with short fixing times have shorter discussions of phenomenon and the solution is starts earlier. There are many possible reasons for the long discussion of the phenomenon. Among them, we focused on the quality and the length of the first report. Intuitively, if the first report contains little information, the discussion of the phenomenon will be long due to the lack of information. We compared the number of words in the description in the first bug report between bug reports with a short bug fixing time and those with a long bug fixing time. The bug reports for the four projects. The y-axis indicates the conversation length. The box on the left represents bug reports with a short fixing time, and the box on the right represents bug reports with a long fixing time. The average value of the conversation length in bug reports for the Mozilla projects with a long bug fixing time is 12.286, whereas that for the four projects is 9.65. The average value of the conversation length in the bug reports for the Mozilla projects with a short bug fixing time is 6.171, while that for the four projects is 5.395. Looking at this box plot diagram, bug reports with a short bug fixing time show that the conversation length of phenomenon is short and the solution is starts earlier.

We investigated whether the results in Figs. 7 and 8 are statistically meaningful using U-tests. The p-value is low (Table 13). Hence, there is a relationship between the conversation length of phenomenon and the bug fixing time.

Results of RQ5:

“Bug reports with short fixing times have shorter discussions of phenomenon and the solution is starts earlier.”

There are many possible reasons for the long discussion of the phenomenon. Among them, we focused on the quality and the length of the first report. Intuitively, if the first report contains little information, the discussion of the phenomenon will be long due to the lack of information. We compared the number of words in the description in the first bug report between bug reports with a short bug fixing time and those with a long bug fixing time. The bug reports
were divided into two groups with the median length of discussion of the phenomenon. Figure 9 and Fig. 10 show the results. The y-axis represents the number of words in the description in the first bug report. The box on the left denotes bug reports with a short fixing time, while the box on the right denotes bug reports with a long fixing time. The average value of the number of words in the first bug report for the Mozilla projects with a long bug fixing time is 93.563, while that for the bug reports for the four projects is 142.5. The average value of the number of words in the first bug report for the Mozilla projects with a short bug fixing time is 99.516, and that for the four projects is 111.3. Neither the Mozilla projects nor the four projects have a relationship between the initial reports and the length of discussion of the phenomenon. U-tests shows a p-value of 0.417 for the Mozilla projects and a p-value of 0.591 for the four projects, demonstrating that the discussion length of the phenomenon is unrelated to the length of the first report. The discussion length of the phenomenon may be related to the content of information in the first report. Additionally, the difficulty of the bug may be related. Further investigation is needed. In the future, we will clarify what information is needed in the first report.

5. Threats to Validity

Internal validity – This study distinguished functional requirements from defect reports. However, other factors such as bug difficulty and experience of people involved were not taken into account. Bug reports have many factors besides written sentences that must also be considered. Examples include the presence or absence of attached files and readability. Hence, categorization is a threat to internal validity. We discussed the categories used with industry experts in our previous study [8]. However, the authors performed the labeling in this study. We will conduct further research about categories in the future.

External validity – To ensure the external validity, U-tests, which is a type of statistical test, was conducted. The experimental results are specific to the dataset used in this study. It is unclear if other datasets will provide similar results because the number of data points is low. Additionally, there are various kinds of bug reports. For example, some bug reports with comments are written within 10 minutes while others have comments spanning more than 100 days. Moreover, some bug reports are abandoned. Hence, the situation depends on the bug report. There is no confirmation that similar experimental results will be obtained in other situations.

Other – Some bugs are released even if they are resolved quickly. Previous studies have noted this problem [28], [29]. For example, consider bug report #806723 in the Firefox. After being fixed, its integration was prevented by two releases. We investigated this problem. This issue is affected not only by when the bug report was written but also by project specific release engineering. After a bug is fixed, it is often reflected in the latest release. Bugs that are fixed in time to be ready for release will be integrated in the next release. Additionally, projects such as Firefox that have a fixed release period have gaps between the fixing time and integration. In any case, users have to wait until the next release even if the bug is fixed quickly. This period varies, and some bug reports are not integrated after 100 days.

6. Usage

This survey provides useful knowledge contained in the discussion of bug reports. RQ1 demonstrates that the number of sentences for each topic does not affect the bug fixing time. Intentionally writing many topic sentences does not influence the bug fixing time. Additionally, mixed topics do not influence the efficiency.

When writing bug report comments, sentence composition is often neglected. However, RQ5 reveals that bug reports with short discussions of the phenomenon have a short fixing time. Bug reports with a short bug fixing time start to discuss the solution topics early. The length of the first report is irrelevant. The content of information in the first report may make it easier to discuss the solution, reducing the bug fixing time. In addition, since patterns of bug reports with a short bug fixing time often contain causes, the causes should be written if it is known.

7. Related Works

The topic most similar to our work is the quality of bug reports. One study evaluated bug reports at three levels based on completeness, readability, the presence of a bulleted item, an attached file, etc. of the first bug report [7]. They investigated factors influencing the quality of the bug report and developed a tool called CUEZILLA. This tool informs the reporter about missing information in the bug report. Another study predicted the bug fixing time using readability, the number of bug reports submitted on a given day, the success rate of the submitter’s past submissions, the number of changes to the bug severity, number of comments, the number of attached files, and the priority as input [1].

Other papers examine cases where the information provided by the reporter and the information required by the developer conflict [13]. Some methods search for complementing information from missing information in past bug reports [14], [15], while others specialize on bugs affecting usability [16].

Moreover, because the reproducibility of bugs is important. It has been investigated [26]. One study found that the questions in the bug report comments are related to the bug fixing time [30]. When developers ask a question to a reporter, it takes a long time to communicate. Hence, the bug fixing time becomes longer. On the other hand, our research focuses on the whole discussion of phenomenon not just the questions.

Research about discussions in software development is part of the field of requirements analysis [17], [18]. One
study investigated the discussion processes from a request to implementation, binary classification in a time series, and clarification [19]. Then pattern analysis revealed six patterns. Eventually the processes were automated to identify which pattern applied.

In response to these, we analyzed the pattern against the comments of the bug report rather than the request discussion. Furthermore, we used multi-value classification. However, we did not identify specific patterns. Ultimately, we intend to identify concrete patterns.

8. Conclusion and Future Work

In this paper, we analyzed the bug report comments to support efficient solutions for bugs. We used five categories about the content, which were identified in our previous work, to analyze the comments. Herein the comments are labeled manually. The number of sentences in each category did not affect the bug fixing time. Additionally, text segmentation revealed that some bug reports mixed topics in each meaning paragraph. Because mixing topics may confuse developers and affect the bug fixing time, we investigated their relationships. However, mixing topics and bug fixing time are unrelated. Hence, it is meaningless to consider the composition of sentences.

Finally, we analyzed the pattern of the bug report comments. We extracted the features of the bug reports with a short bug fixing time by pattern analysis. Bug reports with a short bug fixing time have a short discussion of phenomenon. Additionally, bug reports with a short bug fixing time tend to talk about the cause.

In the future, we aim to discover specific patterns of discussion to efficiently solve bugs and the content of information in the first report. However, more data must be analyzed. Hence, we plan to expand the data. Future analyses will include not only comments but also other factors. It is possible that other factors such as experience of the reporters, the difficulty of the bug, and readability of sentences contribute to the length of discussion of phenomenon. Among these, the content in the first report is currently of great interest. We are planning to make clear definitions and cross checks the categories.

References

[1] P. Hooimeijer and W. Weimer, “Modeling bug report quality,” in Proceedings of the 25th IEEE/ACM international conference on Automated software engineering, pp.34–43, 2007.
[2] C.-Z. Yang, K.-Y. Chen, W.-C. Kao, and C.-C. Yang, “Improving Severity Prediction on Software Bug Reports using Quality Indicators,” in Proceedings of 5th International Conference on Software Engineering and Service Science, pp.216–219, 2014.
[3] K. Aggarwal, T. Rutgers, F. Timbers, A. Hindle, R. Greiner, and E. Stroulia, “Detecting Duplicate Bug Reports with Software Engineering Domain Knowledge,” in Proceedings of the 22nd International Conference on Software Analysis, Evolution, and Reengineering, pp.211–220, 2015.
[4] A. Lazar, S. Ritchey, and B. Sharif, “Improving the Accuracy of Duplicate Bug Report Detection using Textual Similarity Measures,” in Proceedings of the 11th Working Conference on Mining Software Repositories, pp.308–311, 2014.
[5] S. Rastkar, G.C. Murphy, and G. Murray, “Automatic Summarization of Bug Reports,” in Journal IEEE Trans. Softw. Eng., vol.40, no.4, pp.366–380, 2014.
[6] S. Mani, R. Catherine, V.S. Sinha, and A. Dubey, “AUSUM: Approach for Unsupervised Bug Report SUMmarization,” in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, Article no.11, 2012.
[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, “What Makes a Good Bug Report?,” in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp.308–318, 2008.
[8] Y. Noyori, H. Washizaki, Y. Fukazawa, H. Kanuka, K. Ooshima, and R. Tsuchiya, “Improved Searchability of Bug Reports Using Content-based Labeling with Machine Learning of Sentences,” in Proceedings of the 12th Joint Conference on Knowledge-Based Software Engineering, vol.108, pp.75–85, 2018.
[9] M. Galley, K. McKeown, E. Fosler-Lussier, and H. Jing, “Discourse segmentation of multi-party conversation,” in Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, vol.1, pp.562–569, 2003.
[10] M.A. Hearst, “TextTiling: segmenting text into multi-paragraph subtopic passages,” in Journal Computational Linguistics, vol.23, Issue 1, pp.33–64, 1997.
[11] R. Agrawal and R. Srikant, “Mining sequential patterns,” in Proceedings of the Eleventh International Conference on Data Engineering, pp.3–14, 1995.
[12] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in Proceedings of the 20th International Conference on Very Large Data Bases, pp.487–499, 1994.
[13] N.S.M. Yusop, J. Grundy and R. Vasa, “Reporting usability defects: do reporters report what software developers need?,” in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, pp.1–10, 2016.
[14] T. Zhang, J. Chen, H. Jiang, X. Luo, and X. Xia, “Bug Report Enrichment with Application of Automated Fixer Recommendation,” in Proceedings of the 25th International Conference on Program Comprehension, pp.230–240, 2017.
[15] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, “Duplicate Bug Reports Considered Harmful — Really?,” in Proceedings of IEEE International Conference on Software Maintenance, pp.337–345, 2008.
[16] N.S.M. Yusop, J.-G. Schneider, J. Grundy, and R. Vasa, “What Influences Usability Defect Reporting? — A Survey of Software Development Practitioners,” in Proceedings of the 23rd Asia-Pacific Software Engineering Conference, pp.17–24, 2016.
[17] C. Castro-Herrera, J. Cleland-Huang, and B. Mobasher, “Enhancing Stakeholder Profiles to Improve Recommendations in Online Requirements Elicitation,” in Proceedings of the 17th IEEE International Requirements Engineering Conference, pp.37–46, 2009.
[18] M. Rahimi, M. Mirakhorli, and J. Cleland-Huang, “Automated extraction and visualization of quality concerns from requirements specifications,” in Proceedings of the 22nd International Requirements Engineering Conference, pp.253–262, 2014.
[19] E. Knuss, D. Damian, G. Poo-Cuamño, and J. Cleland-Huang, “Detecting and Classifying Patterns of Requirements Clarifications,” in Proceedings of the 20th International Requirements Engineering Conference, pp.251–260, 2012.
[20] https://www.eclipse.org/
[21] https://www.gnome.org
[22] https://www.mozilla.org
[23] https://kde.org/
[24] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, “Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description,” In Proceedings of IEEE International Conference on Software Maintenance and Evolution, pp.125–136, 2017.
[25] K.C. Youm, J. Ahn, J. Kim, and E. Lee, “Bug Localization Based on Code Change Histories and Bug Reports,” In Proceedings of Asia-Pacific Software Engineering Conference, pp.190–197, 2015.

[26] M.E. Joorabchi, M. Mirzaaghaei, and A. Mesbah, “Works for me! characterizing non-reproducible bug reports,” In Proceedings of the Working Conference on Mining Software repository, pp.62–71, 2014.

[27] Y. Noyori, H. Washizaki, Y. Fukazawa, K. Ooshima, H. Kanuka, S. Nojiri, and R. Tsuchiya, “What are good discussions within bug report comments for shortening bug fixing time?,” In Proceedings of the 19th IEEE International Conference on Software Quality, Reliability & Security, pp.280–287, 2019.

[28] D.A. da Costa, S. McIntosh, U. Kulesza, A.E. Hassan, and S.L. Abebe, “An empirical study of the integration time of fixed issues,” In Empirical Software Engineering, vol.23, no.1, pp.334–383, 2018.

[29] M. Choekkertikul, H.K. Dam, T. Tran, and A. Ghose, “Predicting the delay of issues with due dates in software projects,” In Empirical Software Engineering, vol.22, no.3, pp.1223–1263, 2017.

[30] Y. Huang, D.A. da Costa, F. Zhang, and Y. Zou, “An empirical study on the issue reports with questions raised during the issue resolving process,” In Empirical Software Engineering, vol.24, no.2, pp.718–750, 2019.

Yuki Noyori received his B.E. degree in Computer Science and Engineering from Waseda University, Tokyo, Japan in 2018. His research interests include software engineering with machine learning and natural language processing.

Hironori Washizaki is a Professor at the Global Software Engineering Laboratory, Waseda University, Japan. He also works at the National Institute of Informatics as a Visiting Professor, at SYSTEM INFORMATION as an outside director, and at eXmotion as an outside director. He is a technical advisor of GAIO TECHNOLOGY. He obtained his Doctor’s degree in Information and Computer Science from Waseda University in 2003. He is the director of enPiT-Pro Smart SE, chair of IEEE Computer Society Professional & Educational Activities Board Engineering Discipline Committee, chair of IEEE CS Japan Chapter, chair of SEMAT Japan Chapter, convenor of ISO/IEC/JTC1 SC7/WG20, and steering committee members of CSEE&T and APSEC.

Yoshiaki Fukazawa received his B.E, M.E., and D.E. degrees in Electrical Engineering from Waseda University, Tokyo, Japan in 1976, 1978, and 1986, respectively. He is currently a professor of the Department of Information and Computer Science, Waseda University. Additionally, he is the Director, Institute of Open Source Software, Waseda University. His research interests include software engineering, especially reuse of object-oriented software and agent-based software.

Hideyuki Kanuka received his B.E from Musashi Institute of Technology (currently Tokyo City University) in 2001 and M.E from the Tokyo Institute of Technology in 2003. He is currently a Senior Researcher of the Systems Innovation Center at Hitachi.

Keishi Oshima is a Senior Researcher of the Systems Innovation Center at Hitachi. He received his Masters degree in Information Science and Technology from WASEDA University in 2002. His research interests center on legacy system analysis and repository mining.

Shuhei Nojiri is a Senior Researcher of the Systems Innovation Center at Hitachi. He received his M.E. degree in Information Processing and Computer Sciences from Chiba University in 2006. His research interests include software engineering, especially software analysis and management environments.

Ryosuke Tsuchiya is a Researcher of the Systems Innovation Center at Hitachi. He received his Masters degree in Computer Science and Engineering from Waseda University in 2015. His research interests center on legacy system analysis and traceability recovery.