How Practitioners Perceive Automated Bug Report Management Techniques

Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, Baowen Xu

Abstract—Bug reports play an important role in the process of debugging and fixing bugs. To reduce the burden of bug report managers and facilitate the process of bug fixing, a great amount of software engineering research has been invested toward automated bug report management techniques. However, the verdict is still open whether such techniques are actually required and applicable outside the domain of theoretical research. To fill this gap, we conducted a survey among 327 practitioners to gain their insights into various categories of automated bug report management techniques. Specifically, we asked the respondents to rate the importance of such techniques and provide the rationale. To get deeper insights into practitioners' perspective, we conducted follow-up interviews with 25 interviewees selected from the survey respondents. Through the survey and the interviews, we gained a better understanding of the perceived usefulness (or its lack) of different categories of automated bug report management techniques. Based on our findings, we summarized some potential research directions in developing techniques to help developers better manage bug reports.

Index Terms—Bug Report, Developer Perception

1 Introduction

Due to the complexity of software systems, bugs are inevitable. Bug fixing is one of the most important tasks during the process of software development and maintenance. In a recent report[1], Tricentis[2] did an analysis of 606 recorded software failures in 2017. They found that those failures alone affected as many as 3.7 billion people, and caused $1.7 trillion in financial losses. Thus, knowing how to effectively and efficiently fix as many bugs as possible is of great help for the development of software projects.

In practice, bug reports have been playing an important role in bug fixing, since they provide specific details such as description, reproducible steps, stack traces to help developers locate and fix defective code [93]. However, due to the practice of accepting bug reports openly on the web, developers often have to handle a large number of bug reports [2]. Take Eclipse[3] as an example, this project received a total of 514,755 bug reports from October 2001 to April 2017, with an average of 91 new bug reports submitted to Eclipse every day. Meanwhile, since bug reporters may vary in reporting experience, not all bug reports contain sufficient information to help developers fix bugs [93].

To help developers better manage bug reports, researchers have been working on developing different bug report management techniques, such as bug assignee recommendation [2], [31], bug localization [44], [35] and duplicate bug detection [84], [71]. However, it is not clear whether these techniques are perceived as important or whether they truly address practitioners' gripes and challenges.

In this paper, we conducted a survey and follow-up interviews to better understand how practitioners[4] perceive different kinds of automated bug report management techniques. We first conducted a literature review to check all relevant studies published in 17 conferences and journals of software engineering. In total, we summarized ten categories of automated bug report management techniques namely, bug localization, bug assignment, bug categorization, duplicate/similar bug detection, bug report completion/refinement, bug-commit linking, bug report summarization/visualization, bug fixing time prediction, bug severity/priority prediction and re-opened bug prediction. Then we distributed our survey to practitioners with various backgrounds from both industry and OSS communities to provide feedback on these categories. We received a total of 327 responses. Specifically, we invited the respondents to rate the importance of the ten categories of bug report management techniques from very important to very unimportant. We also asked them to provide rationale for a randomly chosen subset of their ratings. Lastly, we conducted interviews with 25 survey respondents to better understand their perspectives. The following are our contributions:

- *Weiqin Zou, Zhenyu Chen and Baowen Xu are with State Key Laboratory for Novel Software Technology, Nanjing University, China. E-mail: wqzou@smail.nju.edu.cn, zychen@nju.edu.cn, bwxu@nju.edu.cn*
- *David Lo is with the School of Information Systems, Singapore Management University, Singapore. E-mail: davidlo@smu.edu.sg*
- *Xin Xia is with Faculty of Information Technology, Monash University, Australia. E-mail: xin.xia@monash.edu*
- *Yang Feng is with Department of Informatics, University of California, Irvine, USA. E-mail: yang.feng@uci.edu*
- *Zhenyu Chen and Xin Xia are the corresponding authors.*

1. https://www.tricentis.com/software-fail-watch/
2. https://www.tricentis.com/
3. http://eclipse.org/
4. In this paper, we refer to "practitioners" as those who actively engaged in software development either professionally or as an actively pursued interest.

TABLE 1
Sources of Papers Reviewed.

Software Engineering Journals/Conferences	Acronym
ACM Transactions on Software Engineering Methodology	TOSEM
IEEE Transactions on Software Engineering	TSE
Empirical Software Engineering	EMSE
Automated Software Engineering	ASE
Journal of Systems and Software	JSS
Information and Software Technology	IST
IEEE Transactions on Reliability	TR
ACM SIGSOFT Symposium on the Foundation of Software Engineering	FSE
International Conference on Software Engineering	ICSE
International Conference on Automated Software Engineering	ASE
International Conference on Program Comprehension	ICPC
International Conference on Software Maintenance and Evolution	ICSME
International Symposium on Software Testing and Analysis	ISSTA
International Conference on Software Analysis, Evolution, and Reengineering	SANER
International Symposium on Empirical Software Engineering and Measurement	ESEM
International Conference on Software Testing, Verification and Validation	ICST
International Conference on Mining Software Repositories	MSR

- We summarize existing bug report management techniques into ten major categories by performing a literature review of papers published between 2006-2017 in 17 conferences and journals.
- We investigate how 327 practitioners from diverse backgrounds perceive the value of different categories of automated bug report management techniques.
- We highlight the rationale behind participants' ratings of different categories of automated bug report management techniques and present some potential improvements for specific automated bug report management techniques.

The remainder of this paper is structured as follows: In Section 2, we describe our research methodology. In Section 3, we present the results of our study. We discuss the insights and threats to the validity of our work in Section 4. In the last two sections, we introduce the related work and conclude our study.

2 Research Methodology

To understand how practitioners perceive research on automated bug report management, we followed a three-step approach. First, we performed a literature review to identify and categorize specific automated bug report management techniques that were proposed by researchers. Next, we designed and distributed a survey based on the categories created at the end of the literature review. Lastly, we conducted interviews to better understand the practitioners' perspectives.

2.1 Literature Review

Our literature review included two parts, paper selection and paper categorization.

Paper Selection: To identify related papers, we first picked a set of 10 conferences and 7 journals, as shown in Table 1. These are major software engineering conferences where papers on bug report managements are likely to be published, and these journals are impactful software engineering journals (with impact factor > 2.0). Besides, these 17 venues have also been used as sources for other software engineering literature reviews, e.g. [66], [41], [90].

Next, we reviewed research papers published in these conferences/journals between 2006-2017. We chose 2006 as the starting point mainly because the research area took off after Anvik et al.'s work on bug assignment that was published in ICSE 2006 [2]. The first and the fifth authors were responsible for extracting relevant papers from the above-mentioned 17 venues. Specifically, these 17 venues were randomly divided into two groups containing 8 and 9 venues respectively. Then each of the two researchers randomly selected one group to work on. To help researchers better select relevant papers, following [39], we defined several inclusion and exclusion criteria. For each conference proceeding or journal, papers were checked against these filtering criteria.

Inclusion criteria:

- The main objective of the paper was to propose automated techniques (e.g. building tools) which help to handle bug reports.
- The paper described the methodology and corresponding evaluation results.

Exclusion criteria:

- Pure empirical studies on bug reports (these studies mainly focus on exploring characteristics of bug reports rather than proposing approaches to help automatically manage bug reports).
- Conference paper versions of later journal articles (if a conference paper was extended to a journal paper, the more complete journal paper was kept).
- Short papers.

To determine whether a paper meets the above criteria, the researchers responsible for certain venues read each paper's title and abstract. They would further browse the main content of those papers when they cannot determine the relevance by merely reading the titles and abstracts. After each researcher tabulated a list of relevant papers for their assigned 8 or 9 venues, they cross-checked each other's paper list, and resolved disagreements through discussions. At the end of the process, we identified 121 papers which focused on automatically managing bug reports from ICSE, FSE, ASE, ICSME, MSR, SANER (WCRE, CSMR-WCRE), ESEM, ICPC, ISSTA, ICST conferences and TOSEM, TSE, EMSE, ASE, JSS, IST, TR journals. The distribution of papers across these conferences and journals is shown in Table 2.

Paper Categorization: Following [36], [37], we performed open card sorting [67] to get the categories of automated bug report management techniques. First, we created one card for each of the 121 papers. Next, the first and fifth authors worked together to identify categories of these papers in two iterations.

Iteration 1: The two authors randomly chose 30 papers, and discussed the categories of these papers. The resultant classification scheme contains 10 categories shown in Table 2. During this process, we found that one paper by Zhang et

TABLE 2
Categories of Automated Bug Report Management Techniques.

ID	Category	Description	Venue	Total
B1	Bug localization	These techniques process a bug report, and locate relevant source code files or methods that possibly contain the bug. Some of these techniques also recommend candidate repairs.	ICSME (6), TSE (3), ASE (3), WCRE (3), MSR (3), FSE (2), IST (2), ICSE (1), ICPC (1)	24
B2	Bug assignment	These techniques process a bug report, and recommend the most appropriate developers to fix the bug.	JSS (5), ICSE (3), MSR (3), FSE (2), ICSM (2), TOSEM (1), TSE (1), EMSE (1), IST (1), ESEM (1), ICPC (1), WCRE (1)	22
B3	Bug categorization	These techniques process a bug report, and classify it into different categories (e.g. reproducible bug report or not, invalid bug report or not, bug fixing request or feature request, security bug report or not etc.)	MSR (3), ASE (2), ICSE (1), ICPC (1), WCRE (1), ESEM (1), EMSE (1), ASE Journal (1), IST (1)	12
B4	Duplicate/Similar bug detection	These techniques detect duplicate/similar bug reports in bug repositories.	ICSE(3), ASE (2), SANER (2), MSR (2), ICSME (1), WCRE (1), EMSE (1), JSS (1), IST (1)	14
B5	Bug report completion/refinement	These techniques aim to generate a high-quality bug report. Some of these techniques automatically generate a bug report when software crashes. Some others help to make a better-quality bug report by enriching/modifying an existing one.	FSE (2), ICPC (2), TSE (1), ICST (1), IST (1), TR (1)	8
B6	Bug-Commit linking	These techniques aim to link bug reports with bug fixing commits or bug inducing commits. With these techniques, developers can better understand which commits fix the bug and why/how/when the bug is introduced.	FSE (3), ICPC (2), ASE (1), WCRE (1)	7
B7	Bug report summarization/visualization	These techniques process a bug report, and summarize it into a much shorter form. Some of these techniques also help developers better navigate/understand bug reports through visualization.	FSE (2), ICSE (1), TSE (1), EMSE (1)	5
B8	Bug fixing time prediction	These techniques process a bug report, and predict how long it will take to fix the bug.	EMSE (4), ICSE (2), MSR (2), IST (2)	10
B9	Bug severity/priority prediction	These techniques process a bug report, and predict its severity/priority.	ICSM (2), FSE (1), ASE (1), WCRE (1), EMSE (1), JSS (1), MSR (1)	8
B10	Re-opened bug prediction	These techniques process a closed bug report, and predict whether it is likely to be re-opened.	ICSE (1), WCRE (1), CSMR (1), EMSE (1), ASE Journal (1)	5

al. aimed to address both bug severity prediction and bug assignment problems [89]. In this case, we placed this paper into these two categories separately.

Iteration 2: The two authors then tried to categorize the remaining 91 papers into the 10 categories independently, and they left out the papers which cannot be categorized for a later discussion. Following [24], [19], Fleiss Kappa [23] was used to measure the agreement between the two labelers, and the overall Kappa value was 0.93. This indicated an almost perfect agreement between the labelers. After completing the manual labeling process, the two authors discussed their disagreements to reach a common decision. They also identified the categories for the unlabeled papers. During the discussion, 6 new categories were created for 6 papers that could not be grouped into the 10 categories in Table 2.

These 121 papers were initially grouped into 16 categories. To make each category representative enough, we filtered out those with less than 3 papers. Using this criterion, six categories (each of which has only one paper) that described studies on predicting the latency of bug reporting [78], predicting the number of bug reports [82], generating candidate repairs from bug reports [45], extracting topics from bug reports [30], predicting the level of defect backlog [68] and mining developers' implementation expertise from bug reports [3] were removed. Finally, we got ten categories covering 115 papers. For each of those categories, we created a sentence that describes it. Table 2 shows these 10 categories along with their descriptions. The full details of the papers belonging to each category are available at: http://github.com/SurfGitHub/bugmanaStudy.

2.2 Survey Design

Protocol: The goal of our survey is to collect the developers' perceptions of the various kinds of automated bug report management techniques. We specifically aimed to answer the following three research questions based on the survey results:

- **RQ1: How do practitioners perceive automated bug report management techniques?**
- **RQ2: Which are the highly-rated automated bug report management techniques that practitioners deem important?**
- **RQ3: Why do practitioners consider certain techniques as important or unimportant?**

In order to collect information that can help us answer these research questions, we designed both close-ended and open-ended survey questions, while limiting answer types to numeric, Likert-scale and short free-form text based on the guidelines in [40].

In particular, we designed 10 close-ended questions for the ten categories of automated bug report management techniques. We listed these ten categories and asked respondents to rate the importance of developing each category to help them better manage bug reports. These questions gathered answers for RQ1 and RQ2. To help survey respondents better understand the categories, concise descriptions (Description column in Table 2) for each category were provided. For each category, like in previous studies [7], [46], we provided five options (i.e. very important, important, neutral, unimportant and very unimportant), plus an additional option: "I don't understand/prefer not to answer". The option "I don't understand/prefer not to answer" was provided in case respondents do not understand the categories of techniques based on the provided descriptions.

Next, we randomly sampled a maximum of two techniques that a respondent had rated as important/very important, and up to two techniques that he/she rated as unimportant/very unimportant. For these randomly sampled techniques, we designed corresponding open-ended questions to ask respondents the rationale behind their ratings. These questions gathered answers for RQ3. In the survey, we also asked some demographic questions (e.g. profession, educational levels) so as to better understand our respondents' backgrounds and to further analyze results by groups (e.g. developers, testers etc.).

Respondent Selection: Our goal is to collect a sufficient number of responses from practitioners with diverse backgrounds to better understand how they perceive various automated bug report management techniques. Specifically, following [41], we tried to get enough respondents from both industry and OSS communities. The detailed selection method is described as follows:

- ***For industrial professionals:*** We contacted professionals working in different companies from different countries. When possible, we also asked them to help us distribute our survey to their associates and colleagues. Through this channel of distribution, we were able to reach industrial professionals in Google, Microsoft, Amazon, Facebook, LinkedIn, TP-link, Huawei, Alibaba, Tencent, Cisco, Baidu and many other small to large companies. This strategy helped us better understand how industrial professionals perceive the value of automated bug report management techniques proposed in the research literature.
- ***For OSS active developers:*** We targeted at both active developers in GitHub and those who had reported/resolved many bug reports in Apache, Mozilla and Eclipse (three projects that are commonly used in bug report management research [2], [93], [86]). On one hand, we intensively mined public commit logs of developers in GitHub and randomly selected 1,000 active developers who made more than 2,500 commits before March 2017. We sent survey invitations to them through email addresses mined from commit logs. On the other hand, we sent emails to 774[5] developers who handled (i.e. reported or were assigned to resolve) more than 100 bug reports in Apache (286)[6], Mozilla (237)[7] and Eclipse (251)[8]. This strategy helped us better understand how automated bug report management techniques are perceived in OSS communities.

Data Analysis: After collecting all the survey responses, we first dropped the "I don't understand/prefer not to answer" option that took only a very small proportion of all ratings. We tabulated a total of 3,270 ratings from 327 survey respondents, with each respondent rating 10 categories. Among the 3,270 ratings, only 29 ratings (0.89%) were for the option "I don't understand/prefer not to answer". This to a large extent, indicated that most respondents understood the 10 categories of techniques we studied in this work. Then, we converted the respondents' ratings into Likert score from 1 (very unimportant) to 5 (very important). Next, we computed some statistics to observe how different demographic groups rated the categories. In RQ1, we calculated the proportions of different ratings for the 10 categories by participants with different backgrounds (e.g. roles, expertise, education etc.). We further conducted Fisher's exact test [22] with Bonferroni correction [48] on proportions of "important" or "very important" ratings to explore whether one demographic group tended to rate automated bug report management techniques higher or lower than other groups. In RQ2, we computed the rating details for each category of bug report management techniques and adopted Scott-Knot ESD test [73] to further explore the kind of techniques that ranked higher among groups with different roles and expertise levels. Lastly, we extracted comments made by respondents to explain why some categories of automated bug report management techniques are considered as important/very important or unimportant/very unimportant.

2.3 Interview Design

Protocol: To get deeper insights into the results of the survey, we conducted follow-up interviews with some of the original respondents. We did this by sending invitations to 107 respondents who provided email addresses in the survey or could be reached through personal contacts. 35 of these accepted our interview invitations. Since the interviews were expected to last about 45 minutes to 1 hour and to ensure a sufficient time allocation to each category being discussed, we decided to focus only on 4 categories of bug report management techniques with each interviewee.

5. We performed two rounds of surveys. One was conducted in March 2017 for 1000 active developers on GitHub. The other one was conducted in April 2018 for 774 Apache, Mozilla and Eclipse developers.

6. For the Apache project, we counted bug reports handled by all 526 members of Apache foundation in its JIRA system; 286 members handled more than 100 bug reports before April 2018.

7. For the Mozilla project, we counted bug reports handled by users with user ID ranging from 1 to 200,000 in its Bugzilla system; 237 users reported bugs within the recent half year (since October 2017) and handled more than 100 bug reports before April 2018.

8. For the Eclipse project, we obtained a total of 251 users who reported bugs to its Bugzilla system within the recent half year (since October 2017) and handled more than 100 bug reports before April 2018.

During each interview, we kept to the following process: First, we explained to the interviewee why we were conducting the interview, and which few categories of bug report management techniques we would discuss (more details in Category Assignment below). Then, we discussed each category sequentially.

During each discussion, we explained the category (what it is about, why researchers study it, and its current state) and ensured that they fully understood it before proceeding. In so doing, we reduced the possibility of bias responses resulting from not having a thorough comprehension of the existing research work. Next, we asked them how they perceive the category but this time we allowed the interviewees to elaborate their viewpoints. We also asked auxiliary questions for clarifications, clarity and insights.

Our interviewees were from China and other countries. We interviewed those located in China through WeChat or in person (for participants from Hangzhou); while we interviewed the rest through Skype or Google Hangouts. We followed the methodology used in [65], [1] to decide when to stop interviewing, i.e. stopping interviews when the saturation of the themes is reached. According to Strauss and Corbin [69], if the already collected data is considered sufficient and further data collection does not generate new information, the sampling should be discontinued. Based on this strategy, we stopped our interviews when we achieved saturation of the themes after we interviewed 25 persons. All 25 interviews lasted an average of about 1 hour with median value of 1 hour; and they were audio-recorded and transcribed for later analysis. Table 3 presents the interviewees' basic demographics.

Category Assignment: We adopted the following assignment method: Given 10 categories, i.e. B1, B2, B3, ..., B10, we considered every continuous 4 categories as an assignment unit. Specifically, we took B1-B4 as the first unit, B5-B8 as the second unit, B9, B10, B1, B2 as the third unit. We repeated this step until we got 25 units. After that, we assigned units to interviewees by their ID number, i.e. the i^{th} unit was assigned to the i^{th} interviewee, with i ranging from 1 to 25. In this way, each category of automated bug report management technique would be discussed by 10 (i.e. 25×4÷10) different interviewees. The details for the assignments of categories to interviewees are provided in the last column of Table 3.

TABLE 3
Interviewees' demographics and their corresponding 4 categories of bug report management techniques discussed.

ID	Role	Experience (years)	categories of bug report management techniques
P1	Developer	8	B1, B2, B3, B4
P2	Tester	10	B5, B6, B7, B8
P3	Developer	6	B9, B10, B1, B2
P4	Developer	14	B3, B4, B5, B6
P5	Project Manager	10	B7, B8, B9, B10
P6	Developer	20	B1, B2, B3, B4
P7	Developer	4	B5, B6, B7, B8
P8	Developer	8	B9, B10, B1, B2
P9	Project Manager	12	B3, B4, B5, B6
P10	Developer	6	B7, B8, B9, B10
P11	Tester	6.5	B1, B2, B3, B4
P12	Developer	12	B5, B6, B7, B8
P13	Developer	7	B9, B10, B1, B2
P14	Project Manager	25	B3, B4, B5, B6
P15	Developer	15	B7, B8, B9, B10
P16	Project Manager	10	B1, B2, B3, B4
P17	Developer	8	B5, B6, B7, B8
P18	Developer	9	B9, B10, B1, B2
P19	Developer	15	B3, B4, B5, B6
P20	Developer	11	B7, B8, B9, B10
P21	Developer	20	B1, B2, B3, B4
P22	Tester	15	B5, B6, B7, B8
P23	Developer	4	B9, B10, B1, B2
P24	Developer	10	B3, B4, B5, B6
P25	Developer	7	B7, B8, B9, B10

3 RESULTS

In this section, we describe how practitioners rated categories of automated bug report management techniques that are summarized in Table 2, together with the rationale of their ratings. We mainly consider three research questions:

- **RQ1: How do practitioners perceive automated bug report management techniques?** (Section 3.1)
- **RQ2: Which are the highly-rated automated bug report management techniques that practitioners deem important?** (Section 3.2)
- **RQ3: Why do practitioners consider certain techniques as important or unimportant?** (Section 3.3)

3.1 RQ1: How do practitioners perceive automated bug report management techniques?

In total, we received 327 responses from 30 countries, of which the top two countries are China (with 58.1% responses) and the United States of America (with 11.3% responses). Among the 327 respondents, 299 are professional software developers, testers, or project managers. The other 28 respondents are non-professionals but they have participated in software development for an average of 9.25 years. 164 of the total respondents have participated in open source projects, among whom 143 are professional developers, testers and project managers. The experiences of our 327 respondents vary from 0.2 years to 35 years, with average experience of 9.87 years.

Considering that 58.1% of the responses were from China, we needed to verify that this would not pose a potential threat to our study. To ensure this, we carried out a comparison between rating results of respondents from China (C) and those from outside China (OC). Specifically, we applied Wilcoxon Test [5] and Cliff's Delta Effect Size [15] to the ratings from respondents in the C and OC groups. The results revealed that there indeed existed a difference (the p-value of Wilcoxon Test is $2.2e^{-016}$), but the difference (measured by Cliff's Delta Effect Size) was small (the calculated effect size is 0.296, whose value falls into the range 0.147 - 0.33 which corresponds to a small difference). Besides, we found that both the C and OC groups of participants have the same median value of ratings. Thus, we conclude that the response imbalance based on country of origin would not affect our results too much.

After collecting all responses from 327 respondents, we investigated how participants from various demographic groups rate the 10 categories of automated bug report management techniques. Following [46], [41], we considered the

following demographic groups:

- All respondents (All)
- Respondents who are developers (Dev)
- Respondents who are testers (Test)
- Respondents who are project managers (PM)
- Respondents with low experience, which we define as the 25% with the least experience in years ($<= 4.0$ years in this survey) (ExpLow)
- Respondents with high experience, which we define as the 25% with the most experience in years ($>= 15$ years in this survey) (ExpHigh)
- Respondents with medium experience, i.e. remaining respondents with more than 4.0 but less than 15 years of experience (ExpMed)
- Respondents with advanced degree, i.e. Master's, Ph.D., M.D. (Adv)
- Respondents without advanced degree (NonAdv)
- Respondents who have participated in OSS projects (OS)
- Respondents who are professional software engineers (Prof)

Figure 1 presents the importance of the 10 categories of automated bug report management techniques as perceived by respondents in different demographic groups. Note that 29 ratings (out of 3,270 ratings) for "I don't understand/prefer not to answer" were excluded from the statistics, i.e. only 3,241 ratings were counted. The horizontal axis shows different demographic groups and the vertical axis shows the proportion of different ratings received from a demographic group. For example, in the **All** group, there were a total of 3,241 ratings, of which 31.0%, 37.2%, 21.4%, 7.6% and 2.7% of them were "Very Important", "Important", "Neutral", "Unimportant" and "Very unimportant" ratings respectively.

From the figure we can observe that practitioners in all of the 10 groups picked more "Very Important" and "Important" ratings than "Unimportant" or "Very Unimportant" ratings. More than 60% - 81% respondents of all groups rated the various categories of automated bug report management techniques as "Very Important" or "Important". Only a minority (less than 17.3%) picked "Unimportant" or "Very Unimportant" ratings in the various categories.

> On the whole, the majority (60% – 81%) of practitioners with various backgrounds considered the automated bug report management techniques as important/very important.

In order to better understand whether one demographic group tended to rate higher or lower than other group(s), we further conducted Fisher's exact test [22] with Bonferroni correction [48] on the proportions of "Important" or "Very Important" ratings from different demographic groups. Fisher's exact test is typically used for categorical data and is always conducted between two nominal variables. Specifically, it is used to test whether the relative proportions of one variable are independent of the other variable. Bonferroni correction aims to control the family-wise error rate for multiple comparisons. We conducted Fisher's exact test with Bonferroni correction on four sets of demographic groups, i.e. groups with different roles (Test vs. PM vs. Dev), groups with different experience levels (ExpHigh vs. ExpMed vs. ExpLow), groups with different educational background (Adv vs. NonAdv) and groups contributing to different kinds of projects (OS vs. Prof).

The detailed results are as follows:

- For groups with different roles, we found that among the ratings that developers (Dev) provided, 65.4% of them were either important or very important. The figures were 81.0% and 68.3% for testers (Test) and project managers (PM) respectively. To validate the significance of rating differences among these groups, we first conducted a Fisher's exact test on each of the three possible pairwise comparisons (Test vs. PM, Test vs. Dev and PM vs. Dev). Then we applied the Bonferroni correction for the three tests. After the Fisher's exact test with Bonferroni correction, we found that all three comparisons were significant (`p-value < 0.05/3`). This meant that developers and project managers were more negative about automated bug report management techniques than testers. Meanwhile, project managers were slightly more positive about the importance of these techniques than developers. These differences, to some extent, can be explained by the fact that testers may be more likely to directly handle different kinds of tasks related to bug reports than developers and project managers.

  > Project managers (68.3%) were slightly more positive about automated bug report management techniques than developers (65.4%); and both of them were more negative than testers (81%) towards these techniques.

- For groups with different experience levels, we found that among the ratings that practitioners with high experience (ExpHigh) provided, 60.6% of them were either important or very important. The figures were 77.3% and 67.4% for practitioners with low (ExpLow) and medium (ExpMed) experience respectively. Similarly, we conducted Fisher's exact test to the three possible pairwise comparisons (ExpHigh vs. ExpLow, ExpHigh vs. ExpMed and ExpMed vs. ExpLow). After Bonferroni correction, we found that all three comparisons were significant (`p-value < 0.05/3`). This meant that practitioners with high experience were more negative about the importance of automated bug report management techniques than those with low and median experience. This may be because experienced practitioners can easily do some bug report management tasks manually and have a better understanding of the shortcomings of existing techniques.

  > Experienced practitioners were more negative about automated bug report management techniques (60.6% for ExpHigh) than less experienced ones (67.4% for ExpMed and 77.3% for ExpLow).

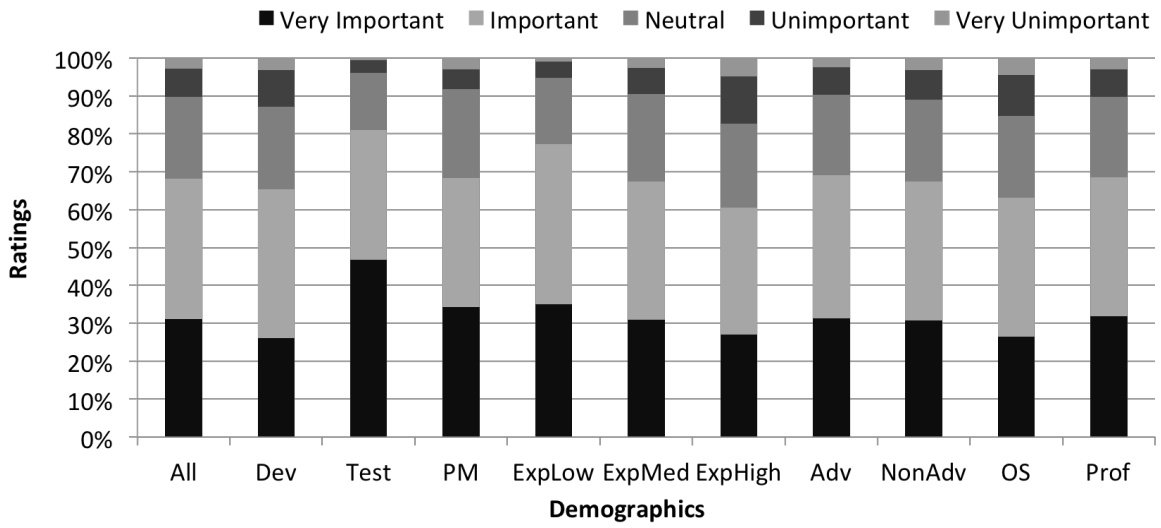

Fig. 1. Importance of 10 categories of automated bug report management techniques to respondents of various demographic groups.

TABLE 4
Detailed ratings from survey responses on automated bug report management techniques.

ID	Bug Report Management Technique	Very Important	Important	Neutral	Unimportant	Very Unimportant
B1	Bug localization	**49.2%**	33.4%	9.6%	6.2%	1.5%
B2	Bug assignment	33.4%	**38.9%**	19.0%	5.8%	2.8%
B3	Bug categorization	33.3%	**41.0%**	17.7%	5.5%	2.4%
B4	Duplicate/Similar bug detection	35.8%	**41.6%**	18.3%	3.4%	0.9%
B5	Bug report completion/refinement	34.5%	**42.8%**	18.8%	3.1%	0.9%
B6	Bug-Commit linking	36.7%	**44.4%**	15.1%	2.8%	0.9%
B7	Bug report summarization/visualization	25.9%	**34.9%**	25.6%	10.8%	2.8%
B8	Bug fixing time prediction	17.8%	28.3%	**31.2%**	15.3%	7.5%
B9	Bug severity/priority prediction	25.6%	**37.3%**	26.9%	8.3%	1.9%
B10	Re-opened bug prediction	17.8%	29.1%	**32.2%**	15.0%	5.9%

- For groups with different educational backgrounds, we found that, among the ratings that practitioners with advanced degree (Adv) provide, 69.0% of them were either important or very important. The figure was 67.3% for practitioners without advanced degree (NonAdv). Again, we conducted Fisher's exact test to validate the significance of the difference. The p-value of difference between groups Adv and Non-Adv was 0.240, which meant the observed difference is not statistically significant (`p-value > 0.05`).

  There was no statistically significant difference between the perceptions of practitioners with advanced degree and those without advanced degree.

- For groups contributing to different kinds of projects, we found that, among the ratings that OSS practitioners provided, 63.2% of them were either important or very important. The figure was 68.5% for professional practitioners. Similarly, we used Fisher's exact test to validate the significance of the difference. The p-value of difference between groups OSS practitioners and professional practitioners was $5.141e^{-05}$, which meant the observed difference is indeed significant (`p-value < 0.05`; Bonferroni correction is not needed for the single test.) This meant that OSS practitioners were more doubtful about the importance of automated bug report management research than professional practitioners. This may be because people have more tolerance of bug reports not being handled in time in OSS community than in the industry.

  Professional practitioners rated automated bug report management techniques higher (68.5%) than OSS practitioners (63.2%).

3.2 RQ2: Which are the highly-rated automated bug report management techniques that practitioners deem important?

Table 4 shows the rating details for each category by 327 respondents. During our calculation, the option "I don't understand/prefer not to answer" was excluded. From the table, we found that for each category, a majority of respondents were positive with it. The proportion of important and very important ratings ranged from 46.1% to 82.6%. Two categories were particularly favored by our respondents; they were bug localization and bug-commit linking. A total of 82.6% respondents rated bug localization as either important or very important. The number was 81.1% for bug-commit linking. The categories receiving the lowest percentages of important and very important ratings

TABLE 5
Ranks of ten categories of automated bug report management techniques according to the Scott-Knott ESD tests for all respondents.

Group	Category of Bug Report Management Technique
1	B1 (Bug localization) B4 (Duplicate/Similar bug detection) B5 (Bug report completion/refinement) B6 (Bug-Commit linking)
2	B2 (Bug assignment) B3 (Bug categorization)
3	B7 (Bug report summarization/visualization) B9 (Bug severity/priority prediction)
4	B8 (Bug fixing time prediction) B10 (Re-opened bug prediction)

TABLE 6
Ranks of ten categories of automated bug report management techniques according to the Scott-Knott ESD tests for developers, testers and project managers respectively.

Group	Category of Bug Report Management Technique
Developers	
1	B1, B4, B5, B6
2	B2, B3
3	B7, B9
4	B8, B10
Testers	
1	B1, B2, B3, B5, B6, B7, B9
2	B4, B8, B10
Project Managers	
1	B1, B2, B3, B4, B5, B6, B7, B9
2	B8, B10

were bug fixing time prediction and re-opened bug prediction. The percentages were 46.1% and 46.9% respectively.

On the whole, a majority of respondents were positive about each category of bug report management techniques. B1 (Bug Localization) and B6 (Bug-Commit Linking) were particularly favored by practitioners (with 82.6% and 81.1% ratings respectively).

In order to gain a better insight into what categories of bug report management research were highly rated by practitioners, we further used Scott-Knott Effect Size Difference (ESD) [73] to group techniques into statistically distinct ranks based on their Likert scores. As a variant of Scott-Knott test [62], Scott-Knott ESD test aims to merge two groups whose difference has a negligible effect size into one group. As highlighted by [73], the Scott-Knott test assumes that data is normally distributed, and it may create groups that are *trivially different* from one another. To address the limitations of Scott-Knott test, Tantithamthavorn et al. proposed the Scott-Knott Effect Size Difference (ESD) test to correct the non-normal distribution of an input dataset and merge any two statistically distinct groups whose difference has a negligible effect size into one group [73].

Similar to Section 3.1, we identified highly rated automated bug report management techniques across different demographic groups, i.e. all respondents, respondents who are developers, testers and project managers, respondents with low experience, medium experience and high experience respectively.

Table 5 presents the 10 categories of techniques as ranked according to the Scott-Knott ESD test for all respondents. We found that four categories of bug report management technique, i.e. B1 (bug localization), B4 (duplicate/similar bug detection), B5 (bug report completion/refinement) and B6 (bug-commit linking) belonged to the first group. This meant that B1, B4, B5 and B6 were ranked substantially higher than other categories. Two categories, namely B8 (bug fixing time prediction) and B10 (re-opened bug prediction) belonged to the last group, i.e. they were given substantially lower ratings as compared to other categories by respondents.

For all respondents, B1, B4, B5 and B6 were ranked highest among all 10 categories; while B8 and B10 were ranked lowest.

Table 6 presents the 10 categories of techniques as ranked according to the Scott-Knott ESD tests for developers, testers and project managers respectively. We found that both B10 (re-opened bug prediction) and B8 (bug fixing time prediction) were ranked lower than other techniques by developers, testers and project managers. The top 3 most ranked techniques among developers, testers and project managers were B1 (bug localization), B5 (bug report completion/refinement) and B6 (bug-commit linking). Additionally, we observed that different from testers, developers and project managers considered B4 (duplicate/similar bug detection) more important than other categories; on the other hand, different from developers, testers and project managers thought more highly of B2 (bug assignment), B3 (bug categorization), B7 (bug report summarization/visualization) and B9 (bug severity/priority prediction).

Testers and project managers thought more highly of B2, B3, B7 and B9 than developers; while developers and project managers ranked B4 higher than testers. B1, B5 and B6 were ranked highest while B8 and B10 were ranked lowest across all roles respectively.

Table 7 presents the 10 categories of techniques as ranked according to the Scott-Knott ESD tests for respondents with low, medium and high experience respectively. Similar to

TABLE 7
Ranks of ten categories of automated bug report management techniques according to the Scott-Knott ESD tests for respondents with low, medium and high experience respectively.

Group	Category of Bug Report Management Technique
Low Experience	
1	B1
2	B2, B3, B4, B5, B6, B9
3	B7, B8, B10
Medium Experience	
1	B1, B3, B4, B5, B6
2	B2, B9
3	B7
4	B8, B10
High Experience	
1	B4, B5, B6
2	B1, B2, B3
3	B7,B9
4	B8, B10

Table 5 and 6, B8 (bug fixing time prediction) and B10 (reopened bug prediction) were ranked lowest by respondents with different experience levels. From the table, we found that different from practitioners with low experience, both practitioners with median and high experience thought more highly of B4 (duplicate/similar bug detection), B5 (bug report completion/refinement) and B6 (bug-commit linking). Moreover, we noticed that respondents of high experience appreciated B1 (bug localization) less than respondents of low or median experience. Perhaps high experienced practitioners may locate bugs more easily by themselves than less experienced ones.

> Practitioners of high experience rated B1 lower than those who are less experienced; while practitioners of low experience appreciated B4, B5 and B6 less than those who are more experienced. B8 and B10 were ranked lowest across all experience levels.

3.3 RQ3: Why do practitioners consider certain techniques as important or unimportant?

To better understand why certain techniques are deemed as important or unimportant by practitioners, we conducted a thematic analysis on the answers/comments collected from the survey respondents and interviewees. Thematic analysis is a technique that aims to identify and record "themes" (i.e. patterns) within textual documents [17], [9]. It generally includes 5 steps: (1) reading through the documents to get familiar with the data, (2) generating initial codes for the data, (3) searching potential themes among the codes, (4) reviewing and refining the themes and (5) defining and naming the final themes. Thematic analysis has been performed in many past software engineering studies, e.g. [16], [64].

Before conducting the thematic analysis, we first did some preprocessing to both the survey and interview data. For the survey part, we carefully read all the comments from the 327 respondents and manually removed comments which do not describe any rationale. For the interview part, we transcribed each audio record into text, segmented the text into units and removed units unrelated with the 10 categories of bug report management techniques. After that, we placed both the survey and interview data together and split them into 10 parts, with each part only containing comments that are related to a certain category of techniques.

In total, 4 researchers, including two non-authors, were involved in the thematic analysis on 10 categories of bug report management techniques. For each category of bug report management techniques, steps (1) to (4) were carried out by two researchers separately. Then, they discussed and resolved conflicts in defining and naming the final themes (step (5)).

The following subsections 3.3.1 to 3.3.10 present the results of thematic analysis on each category of bug report management techniques respectively. The results can be briefly divided into three parts: (1) Reasons for important/very important ratings, (2) Reasons for unimportant/very unimportant ratings and (3) Improvements and Expectations mentioned by some practitioners. To help people better identify relevant themes, in this paper, we use ✓ and ✗ to represent the reasons for important/very important ratings and unimportant/very unimportant ratings respectively.

3.3.1 Bug Localization (B1)

(1) Reasons for important / very important ratings

After analyzing the data, we identified two kinds of rationale for why practitioners considered bug localization techniques important/very important, i.e. speeding up bug fixing and helping novices become familiar with software projects.

✓**Speed up bug fixing:** Eight interviewees and twelve survey respondents thought that a reliable bug localization technique was useful since it could speed up the process of bug fixing. Several explanations were provided by them as follows:

- **Narrow down search space.** As one survey respondent mentioned, *"Any automated process that can help with the debugging process, particularly through static code analysis, would be very useful. Even if it only narrowed down the search area, if it was reliable it could significantly speed up locating the cause of a bug" (Survey).* And if you can find the bug more quickly, *"you can fix it more quickly, then you can work more efficiently."(P21)*
- **Find buggy code owner.** One interviewee explained that with the help of this technique, they could find developers who knew the buggy code best to help them solve the bug more quickly. *"It is more difficult to fix a bug within code that was written by others. This technique can help you find the owner of the buggy code and then you can ask him or her for some help when you encounter problems." (P3)*
- **Estimate bug fixing time.** One survey respondent mentioned that automated bug localization could help casual contributors determine whether they could help to fix a bug quickly. *"This would be useful as a casual contributor to see at a glance whether I know the relevant area of the code, since this is a strong signal that I might be able to fix it quickly." (Survey)*
- **Save time spent in reading bug reports.** Some interviewees and survey respondents said that reliable bug localization techniques could help save the time developers spent in reading bug reports. As a survey respondent commented, *"It would also be useful as a frequent contributor to be able to skim or skip parts of the report." (Survey)*

✓**Familiarize new developers with the code:** Besides speeding up bug fixing, three interviewees commented that another extra advantage of a bug localization technique was that it could help novice engineers get more familiar with the files of code especially working in a big software project. As P21 stated, *"If you have a big complicated software project that has thousands of files like Mozilla or like Wikipedia. And a new developer is not familiar with all of them. They cannot read all the thousands of files on the first day of their work. So if there was some automatic system that helps them find where the bug is, it helps them become familiar with the different files of code. I was a novice engineer, I remember how it was. If there was something*

that shows me around and helps you become familiar with the code. I would really appreciate it."

(2) Reasons for unimportant / very unimportant ratings

Two kinds of rationale were provided by practitioners for their unimportant/very unimportant ratings, i.e. they doubted the reliability of automated bug localization and they did not have a big need for such techniques.

✗**Questionable reliability:** Six interviewees and five survey respondents were skeptical about the feasibility in building effective bug localization tools. They gave various reasons for their doubts:

- **Bugs have different characteristics.** Some thought that different bugs from different domains had different characteristics that could not always be located. As P18 stated, *"Bugs from different domains have different characteristics. I do not think these bugs can be well located by bug localization techniques based on specific algorithms or certain patterns. Sometimes, an inaccurate localization would mislead us and take us more time to fix the bug."*
- **Existing techniques may only work on simple bugs.** Some of them felt that automated tools might only work for simple bugs. One interviewee said, *"It seems that existing techniques mainly make use of the textual similarity between bug reports and source code files to perform bug localization. However, I encountered many bugs that have very little similarity between their bug reports and code files. I wonder what kind of bugs such techniques can localize? Maybe only simple bugs?" (P3)*
- **Different people may describe a bug in different ways.** Some others commented that a bug with the same root cause might be described in different ways by different people, which they felt no tools could locate.
  "Different people will use different ways to describe a bug, how can tools understand that they are actually the same one?" (P23)

✗**Unnecessary and potentially harmful:** One interviewee said he did not need such techniques since he was maintaining a very stable product with only simple bugs to fix. *"Our major work is to maintain a well-operated system which has been running for more than 20 years. We mostly conduct minor changes to our system and most bugs we encountered are very easy to fix." (P1)*

Another interviewee shared his concerns that these techniques might hinder the further growth of novice developers. *"Manually debugging bugs is an essential step for a novice developer to become an experienced one. I think a developer with a smart mind is more valuable than a novice person equipped with an AI like tool. Automatically helping a fresh hand locate bugs is bad for him or her in the long run." (P16)*

(3) Improvements and Expectations

Incorporate additional relevant information: Eight interviewees mentioned that bug screenshots, stack traces, recorded logs and videos were very useful in locating bugs. Some of them suggested making full use of these kinds of information to improve the performance of bug localization techniques. The relevant comments are as follows:

"When we locate bugs, we not only read bug reports but also read screenshots, logs and stack traces that play a key role in bug localization. You should try to combine these kinds of information together so that you may get a workable tool." (P6)
"Besides screenshots, sometimes our bug reporters would also upload some videos to show what bugs they are exactly facing. Those videos are really helpful in locating bugs." (P23)

Find hard-to-locate bugs: Seven interviewees expressed a great need for techniques that can help them locate specific kinds of bugs such as performance bugs, memory leak bugs, environment-related bugs caused by hardware crash or bad network etc. Those bugs, as they said, were always hard to locate. Some of their comments are as follows:

"Our platform includes some third-party or open source components. For those bugs occurred in these components, we always need to spend much time sometimes even more than a 1 week to fix them. Besides, memory leak bugs are also difficult to locate." (P16)
"In my experience, two kinds of bugs are very hard to locate: performance bugs and environment-related bugs caused by such as bad network or hardware crash. It would be fantastic if some kind of techniques can help us locate these bugs." (P13)
"It would be great if such techniques can help us locate bugs that hide in code with complex logic." (P8)
"We often encountered random bugs that were introduced by using some encryption and decryption algorithms. Locating them is a difficult task." (P25)

3.3.2 Bug Assignment (B2)

(1) Reasons for important / very important ratings

Three kinds of rationale were identified during our thematic analysis, i.e. speeding up bug fixing, saving triager's valuable time and improving the visibility of unnoticed bug reports.

✓**Speed up bug fixing:** Seven interviewees and eight survey respondents mentioned that bug assignment techniques could help bug reports to be resolved in a faster way by helping to direct them to suitable developers. In this way, much unnecessary communication within/across teams or departments could be avoided. And this could further *"happy customers if a bug is resolved asap and reduce the loss caused to an organization due to bugs." (Survey)*

"Making it clear what the problem is and making it clear who is responsible for handling (or further diagnosing) the bug is the most critical part of getting a bug fixed." (Survey)
"Wrongly assigned bug reports will always face tossing between different developers within or even across teams or departments. Such tossing is very time-consuming and would drive us crazy if these bugs have high priority but cannot be fixed in time." (P16)
"Remove some routine operations that delay the time before real work starts." (Survey)

✓**Save triager's valuable time:** Five interviewees and three survey respondents said this technique could help

save much manual work in assigning bugs. As some interviewees explained, only experienced people (who are very familiar with the system) are qualified for bug assignment. Automated bug assignment tools can save their precious time and allow them to do other more value-added tasks.

"In our company, only experienced developers can be appointed as bug triagers. They usually spend a lot of time to ensure that each bug goes to the right fixer. So, if you can help them automatically identify qualified fixers, they would be very happy since they can have more time to do other important tasks." (P6)
"Developers often neglect triage because it's time-consuming and not as rewarding as development / fixing the bugs." (Survey)

✓**Improve bug report visibility:** Six survey respondents mentioned that automated bug assignment let developers realize the existence of some bugs that were not noticed before. One survey respondent said that a lot of bugs reported to many open source projects (which have many bugs coming in) go untriaged for a long time.

"The important part of triaging to me would be, for example, to automatically check if a bug still persists or if it has been fixed in the meantime without anyone realizing." (Survey)
"Bugs which the developers are not made aware of are less likely to be noticed and fixed." (Survey)
"Many open source projects have a high volume of bugs coming in and a lot of them can go untriaged for a long time." (Survey)

(2) Reasons for unimportant / very unimportant ratings

After analyzing the data, we found that some practitioners thought automated bug assignment techniques to be unimportant because they doubted the performance of such techniques or they did not have such a need.

✗**Questionable reliability:** Two interviewees and two survey respondent were skeptical about the effectiveness of automated bug assignment techniques. They thought that bug assignment depended a lot on the knowledge a developer had. One interviewee doubted whether existing techniques could properly assign bugs of same root cause but with different descriptions properly.

"I can't imagine it would work very well without most of the knowledge one of the project's developers has in their head." (Survey)
"Existing bug assignment techniques seem to be mainly based on textual similarity, how can it handle those bugs that share the same root cause but described in different ways?" (P23)

✗**Unnecessary:** Five interviewees and three survey respondents commented that they did not need such a technique either because they could do it manually, or they did not have an enforced assignment process.

- **Manual assignment is sufficient.** Four interviewees and two survey respondents said that they did not need such a tool since they could easily assign bug reports to suitable developers.
  "Usually it is not hard to find the right contact" (Survey)
  "Our development is in a quick way. Each version iteration would last no more than one month, during which only few bugs will manifest. Both our developers and QAs are familiar with modules thus bugs can always be assigned properly." (P13)
- **There is no assignment process.** One interviewee and one survey respondent said that they did not have an assignment process, instead, the developers took on the work by themselves.
  "Some teams practice agile methodology, scrum. So if you work according to the agile scrum methodology then, usually developers themselves just decided, I will take this one and you will take this bug and we'll share this. That's how it usually works with agile or scrum. No assignment process. They just take the work, people just take the work themselves." (P21)
  "For the most part though, the core team contributors decide on their own what to work on. Our model is based on trust and autonomy and has very little central control." (Survey)

(3) Improvements and Expectations

Incorporate additional relevant information: In terms of feasibility in building effective bug assignment tools, seven interviewees thought that bug assignment tools could not work effectively without considering other sources of information in addition to the text in the bug reports, – e.g. developers' expertise and their current state (i.e. availability etc.) etc.

"Effective bug assignment requests a good understanding of the system, involved developers (e.g. whether they are busy or not, tasks at hand) and bugs' effect on business. How can a bug assignment technique work without complementing such context information? " (P18, P3)
"I think it will greatly improve the performance of bug assignment techniques if you also consider the root cause of bugs and developers' expertise (i.e. who are good at solving which kinds of bugs (e.g. memory leak, performance, UI and workflow problems))." (P11)

Associate bugs to their relevant components: Eight interviewees said that their bugs would be first assigned based on components, then re-assigned to the components' developers. Some of them desired some tools that could help them correctly associate bugs with all relevant components. In this way, they could easily find right developers related to the components to work out the solution for a bug.

"Whenever we received a bug report, we would first figure out which component this bug belongs to, then assign it to the developers who are currently responsible for the component." (P1)
"We often have bugs that are related to more than one component. It's pretty convenient if you can associate bugs with several components. This helps people who have expertise in different components and different topics to work together and work out a solution." (P21)

Assign a bug to a pair of developers: One interviewee suggested assigning each bug to a pair of developers (i.e.

a senior one and a novice one) rather than a single proper fixer. The senior developer guides the novice developer to fix the bug. He said this strategy works well in his company.

"In our company, two developers are responsible for fixing a bug – a senior experienced developer aims to roughly figure out the potential root cause of the bug and a novice one aims to fix the bug under the guidance. This practice is good since it can ensure the development progress and help novice developers obtain an overall view of the system." (P8)

Collaborate with human: One interviewee thought it would be good if tools could correctly assign as many *simple* bugs as possible and leave only complex bugs to humans to handle. Another interviewee thought an automated bug assignment system should have the ability to improve itself through interactions with humans.

"I think it may be more practical to, e.g. try to have most simple bugs effectively assigned by tools and only leave complex bugs which cannot be handled by tools to the triagers." (P6)
"I think a good bug assignment system should be able to receive feedbacks from people and get improved through learning." (P23)

3.3.3 Bug Categorization (B3)

(1) Reasons for important / very important ratings

After thematic analysis, we identified the following three kinds of reasons why practitioners deemed automated bug categorization as important/very important:

✓**Better resource allocation:** Eight interviewees and ten survey respondents mentioned that automated bug categorization techniques could help people better allocate resources. With such techniques, developers can easily filter bug reports to work on and thus get a higher throughput in bug fixing.
"I am a maintainer of an open source project with about a million users and hundreds of contributors. I've become a bottleneck in categorizing bug reports and assigning them to contributors. As a result, contributors don't know what to work on and many bugs don't get fixed. Tooling in this space might help with better resource allocation and higher throughput in fixing bugs." (Survey)

✓**Save manual work in classifying bugs:** Five interviewees and four survey respondents thought this technique can save much manual work in classifying bug reports, especially for those large open source projects or crowdtesting projects.

"You cannot expect a brand new person on the job to just jump into it and start classifying bugs. The responsibility is supposed to be with the project manager and they should have a lot of experience, otherwise they will make a lot of mistakes. This kind of technique will be very useful and important." (P21)
"I think this kind of technique will help large open source projects and crowdtesting projects a lot because they often have many bug reports reported by various users." (P6)

✓**Facilitate postmortem analysis:** Two interviewees and two survey respondents commented that bug classification could help in later analysis. Two kinds of help were mentioned in their comments as follows:

- **Evaluation of product quality.** One interviewee and one survey respondent stated that automated bug reports classification could help them better understand what problems their products have.
  "Sometimes we need to analyze bug reports to figure out what problems our products tend to have. This technique would help a lot when we do such an analysis." (P9)
- **Identification of common bug fix solution.** Two interviewees and one survey respondent thought that classified bug reports could more likely help them find a common solution to a certain kind of bugs.
  "If bugs have been divided, we will likely find a common solution of this kind of bugs." (Survey)

(2) Reasons for unimportant / very unimportant ratings

During analysis, we found that practitioners considered this technique unimportant either because they could manually do it or they were skeptical about the effectiveness of this technique.

✗**Manual work is sufficient:** Two interviewees and three survey respondents said they could manually do bug classification easily. As P11 stated, *"In our company, testers are required to fill in the report whether the bug is valid and reproducible. And they are experienced at doing this. If they cannot determine, which is very rare, our project manager or developers will do them a favor." (P11)*

✗**Questionable reliability:** Two survey respondents expressed their doubt on the reliability of automated bug categorization techniques. As one respondent explained, *"Categories are very quickly devalued (at least subjectively) by even a few miscategorisations. A categorisation algorithm is likely to put too much weight on just the presence of words. Even if a bug report mentions a category name 10 times it does not necessarily mean that it belongs in that category." (Survey)*

(3) Improvements and Expectations

Incorporate information from screenshots: Some interviewees suggested making more use of screenshots to do automated bug categorization. They explained that their testers were often required to attach bug screenshots while reporting bugs and these screenshots helped them a lot in classifying bugs manually.
"It's not hard to classify bug reports when they have screenshots. In our company, testers are requested to provide relevant screenshots while filing a bug report." (P1)

Classify bugs based on other desired categories: Half of the interviewees mentioned that instead of classifying bugs as valid or invalid, severe or non-severe, they would like to see more research work done to classify bugs based on their root cause, potential impact on business, reproducibility and security concerns etc. Some representative comments are as follows:

- **[Root cause and business impact]** *"Is it possible for the research community to classify bugs at a more fine-grained*

level? E.g. by telling us whether the bug is a performance bug or a functional bug, whether the bug will affect a large number of users. This is what we really want." (P24)

- **[Reproducibility]** *"Reproducibility in particular is great, because reproduction is always the longest step in actually fixing a bug" (P9)*
- **[Security concerns]** *"I usually work with very open bug reporting systems that don't have any secrets because it's open source. However, it may sometimes happen that something does have to be secret. For example, some bug reports are about security. The bug report itself should be secret so that bad people won't use the same way then walk into my website without the password. So, if I see that this bug report must be secret, then I will mark it as something that should be secret, so there will be less damage." (P21)*

3.3.4 Duplicate / Similar Bug Detection (B4)

(1) Reasons for important / very important ratings

We identified the following four major reasons respondents/interviewees provided for their important/very important ratings:

✓**Save developers' time:** Eight interviewees and twenty two survey respondents mentioned that duplicate bug detection techniques could save developers' time in identifying duplicate bugs.

"In our team, there is a bug report manager being responsible for reviewing bug reports submitted by different testers. He needs to check a bug report's validity and mark whether it's duplicate or not. This process sometimes is very time-consuming when he has to search and read existing bug reports in our JIRA system. Anything that can help him identify duplicate bugs will do a great help." (P9)
"For any sufficiently large open source project, it becomes very difficult to keep track of all the issues that are open and avoid duplication - it's an N:N problem. For maintainers, marking duplicates can be a time-consuming housekeeping task. Anything that gets me that time back is helpful." (Survey)

✓**Save reporters' time:** Six interviewees and ten survey respondents thought that this technique could help reporters avoid reporting duplicate bug reports by providing potential duplicate reports to them. Some said it was difficult for reporters to find out whether their bugs had been reported or not.

"It's the hard part of bug reporting for both reporter and the developer." (Survey)
"Users filing bugs that are already duplicates of other bugs is a big problem and automatically detecting duplicates during the bug filing process is very helpful." (Survey)

✓**Provide hints for bug fixing:** Four interviewees and six survey respondents mentioned that duplicate bug reports could help fix bugs. Some interviewees said that when they encountered a new bug, they would search bug tracking systems to find whether there were some potential duplicate/similar/relevant bug reports that could provide some fixing hints.

"Whenever I found a previous bug report related to the bug I was fixing, I would check whether its solution could be applied to my bug. Sometimes I would directly refer to its fixer for further help to fix my bug." (P4)

✓**Filter bug reports:** Four interviewees and two survey respondents commented that duplicate/similar bug report detection could help developers choose what bug reports to work on. P21 said, *"As an engineer and product manager, duplicate bug reports are an indication of importance." (P21)*. One survey respondent said, *"with the help of duplicate bug detection techniques, developers can more easily find those non-duplicate bug reports to work on." (Survey)*

(2) Reasons for unimportant / very unimportant ratings

Four interviewees and one survey respondent mentioned the following three reasons why they thought this technique was unimportant/very unimportant:

✗**Duplicate bug reports are rare:** Two interviewees said that they did not need such techniques since they seldom came across duplicate bugs. As P1 stated, *"We seldom encountered duplicate bugs in our project." (P1)*

✗**Manual work is sufficient:** One interviewee and one survey respondent said it was not difficult for people to manually identify duplicate bug reports.

"When duplicate bug reports do happen, I don't think that's a big problem. I don't think it's a big waste of time. It's not very difficult, an engineer or a product manager who is familiar with the project is supposed to identify duplicate bug reports very quickly. " (P21)
"People can find duplicates on their own." (Survey)

✗**Developers directly fix bugs without considering duplicate reports:** One interviewee said they did not care whether a bug had been reported or fixed, instead, they would directly fix it.

"As long as a bug occurs in certain version of our product, we just fix it without considering whether or not it has been reported or fixed in other versions." (P24)

(3) Improvements and Expectations

Identify duplicate bugs with different symptoms: Five interviewees mentioned that it was difficult for them to identify bugs that had different symptoms but had the same root cause. They hoped researchers could help them better identify this kind of duplicate bug reports.

"We do hope some techniques can be developed to help us find those bugs with same root cause but different symptoms." (P6)

Perform more empirical analysis on duplicate bugs: Five interviewees hoped that researchers could do some empirical analysis on duplicate bug reports. They would like to know what kind of bugs tend to be duplicated, the difference between duplicate bugs of projects from different domains (e.g. traditional PC applications vs. mobile applications) etc.

"It would be very helpful if researchers can do some summarizations on duplicate bug reports. We are eager to know the characteristics of bugs that tend to be duplicate. In this way, we can propose suitable actions to cope with these duplicate bugs." (P16)
"I think it is necessary to do some comparisons between duplicate bugs from different domains. Duplicate bugs that occurred within PC products may have different characteristics compared to those occurred within mobile products. These comparisons may provide a good guidance for developers working in different companies." (P19)

3.3.5 Bug report completion / refinement (B5)

(1) Reasons for important / very important ratings

We identified three kinds of rationale practitioners provided for why they deem automated bug report completion/refinement techniques important/very important as follows:

✓**Easier bug fix and reproduction:** Nine interviewees and fourteen survey respondents commented that a complete and correctly-filled bug report was the top prerequisite of successful bug reproducing and fixing. Some of them mentioned that it was unavoidable for reporters to make some mistakes in filing reports especially in a complex system with many components. Any tool that could help them get a better bug report or know that some key items were wrongly filled would be highly appreciated. Some pertinent comments are as follows:

"The more details a bug report gives, the easier it is to reproduce and fix a bug. We sometimes received bug reports whose important information (e.g. component and hardware platform for android apps) was missing or wrongly filled. It would be helpful if some items can be automatically filled; and knowing some items are wrongly filled itself is also important when locating bugs." (P7)
"Some bugs are difficult to reproduce (race conditions, "Heisenbugs" and the like). Any helpful information about the state the system was in when the bug occurred can be vital to locate the root cause." (Survey)
"I have seen in projects like Fedora that there are many component options and they are hard to fill in correctly." (P19)

✓**Better reporting experience:** Seven interviewees and five survey respondents felt that with the help of such tools, the work of bug reporters would be easier as they would not need to collect/fill all necessary items and would feel more relieved when they wrongly filled some items unintentionally. This would motivate them to submit a good bug report.

"Anything that makes it easier to submit a good bug report will make it more likely that people do so." (P14)
"I always made some mistakes when I submit a bug report, e.g. wrongly assigned a value to a field such as platform or version, which will cause the difficulty in bug fixing" (Survey)
"Usually it is a human that collects this information when there is an issue or a crash situation and there is room for avoiding some critical information due to the nature of human. On the other hand there might be information that a tool can collect at the crash which cannot be recovered by any other means later on by a human." (Survey)

✓**Ease of manual analysis:** Two survey respondents mentioned that automatically generated bug reports may have the consistent format, which would make it possible for people to easily do some analysis on them, e.g. to find the possible cause for the same bug by highlighting the commonalities and to know the number of users reporting the same problem.

"Maybe it is possible to collect multiple automatically generated bug reports for the same bug and highlight the commonalities and thus focusing on the possible cause." (Survey)
"...The format will be consistent, which means you can run all sorts of interesting analyses (e.g. how many users reporting the same problem)." (Survey)

(2) Reasons for unimportant / very unimportant ratings

Practitioners who were negative about the development of bug completion/refinement techniques felt so because they did not have such a need or they doubted the quality of an automatically generated bug report.

✗**Unnecessary:** Three interviewees mentioned that bug report completion/refinement techniques were not applicable to their products because their experienced testers were not likely to make mistakes while filling a bug report. Another interviewee declared that their bug reports requested only a bug description, thus they did not need such a technique. Some representative comments made by interviewees are as follows:

"In our company, testers are always familiar with the tested products. We also have strict rules on how to file a bug report. Thus it's almost impossible for testers to submit an incomplete or a wrongly filled bug report." (P2)
"The project I have worked most closely on for the longest time (Interchange e-commerce framework) does not have a component selection – it just asks for a bug description." (P12)

✗**Questionable quality:** Two interviewees and two survey respondents doubted whether it was really possible to automatically generate a workable bug report with detailed description and reliable reproducible steps.

"I can understand that some basic information such as version, hardware platform, operation system can be collected. How is that possible to exactly describe the way to reproduce the bug? Even human cannot do it well sometimes." (P9)

(3) Improvements and Expectations

Place greater emphasis on specific bug report information: Six interviewees highlighted the importance of some items within a bug report, including priority, potential impact on users, reproducible steps, environmental configurations and product/component etc. They hoped bug completion/refinement tools can provide accurate and reliable information in these specific areas.

- **[Priority and potential impact on users]** *"Whether this bug has a high priority and how many users are*

affected are what we really care about." (P24)

- **[Reproducible steps and product/component]** *"I hope an automatically generated bug report can tell us how to reproduce the bug and which product/component it belongs to, then we can quickly find the right person to fix it." (P14)*
- **[Environmental configurations]** *"If the software crashed, you typically really need to know what exactly they're doing, whether they're running windows or whatever, what browser they're using, the version of the browser. Otherwise it's a waste of time. Some normal general users may feel difficulty in filling some items when they report a bug report."(P17)*

Guarantee user privacy: One survey respondent desired an automated bug report generation tool that guarantees user privacy. As he/she stated, *"A lot of my development is in Java, so having a high quality automatically reported bug report that adequately guards privacy and what caused it would be very useful!" (Survey)*

3.3.6 Bug-Commit Linking (B6)

(1) Reasons for important / very important ratings

After analyzing the data, we identified the following five major reasons why practitioners considered bug-commit linking techniques important/very important:

✓**Speed up bug fixing:** Seven interviewees and sixteen survey respondents said that bug-commit linking techniques were very helpful for developers in that they could fix bugs easier by directly referring to bug introducing commits.

"Sometimes it's not that easy to identify relevant commits to bugs especially those that randomly occurred. If we know bug introducing commits, then we can directly go to the commits, study the code changes and propose a solution. Suppose we are not familiar with the product, we still can assign the bug to the commit submitting person to fix it." (P17)
"Being able to move from buggy code back to a discussion about the implementation that originally introduced the bug is one invaluable part of the multi-step process of variable isolation." (Survey)

✓**Improve bug / commit understanding:** Four interviewees and eleven survey respondents thought this technique was helpful since it could help developers better understand why some commits were introduced and what happened to reported bugs. This kind of status tracking was also very essential for release management where *"release managers need to exactly know which bugs will be fixed in each release recycle." (Survey)*

"It's very tricky to force people to do it [the linking], you just have to really encourage it because it's really nice if someone comes in and just happens to pick something...And then it turns out that if they fix it, but nobody knows because the bug database didn't get updated or whatever. So it's tricky. That's more social thing. I feel like every single change you make to the code base, there's a reason for it." (P22)

"It helps determine why some commits were introduced. Especially if a "hack" was needed, this helps understanding the context of some code. You need to know why/what/when and by whom happened to those issues/bugs. Referencing is very important." (Survey)

✓**Save developers' time:** Six interviewees and three survey respondents mentioned that automated bug-commit linking techniques that can do away with manual linking would surely save developers' time. As two interviewees explained, it was time consuming to get things done manually, especially when they had as many as several hundred commits submitted to the codebase everyday.

"It takes a lot of time to manually dig up these cross-links." (Survey)
"It would be very useful if you can link bugs with their inducing commits. But it is too hard for us to manually do this, every day we have several hundred commits submitted to our project." (P24)

✓**Help find preventive measures for similar problems:** Four interviewees and two survey respondents thought that links between bugs and relevant commits could help developers/teams better understand their products (e.g. which parts were error-prone) and find the weaknesses of their development process (e.g. what went wrong with the testing). They would then be able to take some corrective actions to avoid similar problems in the future.

"You don't just have this is fixed by such and such a commit. You want to say this was broken in such a commit because..., then you can work out why didn't our tools catch it? What was wrong with our testing?" (P22)
"Sometimes we would have a long discussion to summarize what problems we have, which parts of our product are error prone and why, whether we can avoid them in the future. Tools that exactly link bugs and relevant commits would help a lot during our analysis." (P14)

✓**Help ensure bug fix correctness:** One interviewee and two survey respondents thought that linking bugs with their fixing commits could help guard the correctness of bug fixing commits, e.g. testers could design better test cases by knowing the fixing points and the third party could do a better code review.

"If testers know the bug fixing points in advance, then they can design more proper test cases to do a better test." (P2)
"Helpful in 3rd party review of fixes." (Survey)

(2) Reasons for unimportant / very unimportant ratings

In terms of necessity of bug-commit linking tools during their own development, four interviewees and four survey respondents did not perceive much value in this kind of technique:

✗**Existing support for linking bugs and bug fixing commits should suffice:** Four interviewees and three survey respondents declared that it was easy to link bugs with bug-fixing commits on GitHub/Gitlab.

"GitHub already provides the function of automatically linking bug fixing commits with bug reports. The only thing you need to do is to add the bug ID in the commit log. E.g. fix #xxx, close #xxx, resolved #xxx. Then the bug report will be automatically closed. It's easy." (P7)

✗**Does not make sense for large commits:** Two interviewees said that linking bugs and commits did not make sense during their development because their commits were always large.

"We do not commit often. Actually, we always submit large commits that contain both feature implementation and bug fixing. Knowing which commits introduced a bug cannot help us narrow down search space while locating bugs. " (P12)

✗**Questionable reliability in complex cases:** One interviewee and one survey respondent were skeptical about the effectiveness of such techniques in complex cases. The interviewee did not believe tools could find bug inducing commits that humans could not. One survey respondent doubted its capability in complex projects.

"At most time when I receive a bug report, I can immediately realize where the problem is. For those I cannot determine, I don't think tools can." (P19)
"In practice bug systems and source control systems don't always line up 1:1 in more complex projects." (Survey)

3.3.7 Bug Summarization / Visualization (B7)

(1) Reasons for important / very important ratings

We identified four major reasons why practitioners deemed bug summarization/visualization techniques as important/very important as follows:

✓**Improve problem understanding:** Three interviewees and three survey respondents thought this kind of technique could help find problems their products tended to have and could further give hints on what to improve.

"Getting a high level visualization of bugs could be useful for understanding where there are problems and what could be done to improve the process." (Survey)

✓**Speed up bug fixing:** Four interviewees and three survey respondents thought this kind of technique could help speed up bug fixing because developers could easily understand the problem without reading unnecessary details or manually summarizing and experienced triagers could do a quicker bug assignment.

"A good bug report summary can help us get a quick overview of the bug report. We do not need to read everything of the report. We also do not need to manually add summaries to the bug reports in our backlog list." (P10)
"For a bug triager who is familiar with the whole system, providing a good short (e.g. 20-words-length) summary is enough for him/her to effectively triage the bug." (P7)

✓**Help bug-fix prioritization:** Two interviewees and two survey respondents mentioned that this kind of technique could help them easily select the next bug report to work on.

"Being able to skim read a list of bug summaries and pick the next on your agenda is usually better and sorting summaries by severity scales only helps the process in a minor way. Being able to cherry-pick the next bug you need to work on quickly from a supply of open ones is the key metric." (Survey)

✓**Aid discovery of customers' potential interests / needs:** As one interviewee mentioned, in their bug tracking system, some bug reports actually were feature requests. Summarizing these feature requests could also help project managers have an additional insight on what customers really wanted and they could further recommend some attractive features to them.

"Both bugs and feature requests are reported in our JIRA system. Project managers would always check the JIRA data to find out what bugs we fixed before and what features our customers usually requested. Thus they can further recommend these attractive features to potential customers. Summarizing these reports can help them a lot while conducting these analyses." (P5)

(2) Reasons for unimportant / very unimportant ratings

Three major reasons were mentioned by practitioners for why they thought it was unimportant to develop bug summarization/visualization techniques.

✗**Hinder bug fixing:** Different from bug triagers or project managers who may value this kind of technique a lot, four interviewees and five survey respondents mentioned that such techniques were not needed by bug fixers since what they wanted were detailed bug reports.

"Bugs are all about the details. Summarizing a bug report seems almost certain to render it useless." (Survey)
"Bug summarization would ignore lots of details that are essential for bug reproducing. I cannot see its values in fixing bugs." (P15).

✗**Unnecessary:** One interviewee and one survey respondent said they did not have such a need for bug report summarization techniques. One explained that they seldom encountered bug reports with too much information. The other one declared that the item of one-line summary of bug reports in JIRA or Bugzilla was already enough for them to briefly understand what the problem was.

"This isn't something I have ever felt a need for. Bug reports with too much information are a rarity and when they occur extracting the key information is usually trivial." (Survey)
"The one-line summary item of a bug report is already a summary of a bug in the JIRA or Bugzilla system. I think it's already good enough for us to understand the problem." (P17)

✗**Questionable reliability:** One interviewee and two survey respondents were skeptical about the effectiveness of bug summarization/visualization techniques.

"Unless the bug is stupid simple, having a summarization wouldn't help and I don't see how a bug report could be visualized unless it has to do with graphics programming, just read the bug." (Survey).

(3) Improvements and Expectations

Include essential information: Five interviewees expressed their opinions on what makes a good bug report summary. They mentioned that a good summarization should contain the essential information that could help developers quickly understand and fix the bug. These information should include for example, **one-line summary, steps to reproduce, potential effect on users, priority/severity, reporter, suggested fixing methods and some key exceptional logs (if they exist)** etc. Some relevant comments are as follows:

"One line summary is so important, because you have to try and get the information as to say what's going on. Um, but I think that and then your priority and severity, so there is how bad is it? How bad is it for your customers? And the priority is now which one are we going to fix next? I think those three are the absolute essentials." (P22)
"It is valuable if you provide a summarization that filters out all unnecessary details but keeps essential items for bug fixers, such as a brief description of a bug, reproducible steps and some key exceptional logs etc." (P20)
"If you could also briefly suggest fixing solutions, that would be very fantastic. Reporter's name should also be included for later contact in bug fixing.' (P25)

Visualize bug dependency: One survey respondent described a tool that he/she thought would be useful, i.e. a tool that tries to *"visualize the dependency graph of bugs that are currently considered must-fix for a specific release and make it easy to add or remove bugs from the list." (Survey)*

3.3.8 Bug Fixing Time Prediction (B8)

(1) Reasons for important / very important ratings

After performing thematic analysis, we identified the following three major reasons why practitioners considered bug fixing time prediction techniques important/very important:

✓**Help make a better overall schedule:** Three interviewees and four survey respondents thought that knowing bug fixing time could help developers themselves make a better schedule for doing different tasks.

"Actually, for many developers, fixing bugs is only a small part of their daily routine. When they are busy, knowing whether the bug is big or not is useful enough for them to decide when to fix it. I mean, they don't need to spend extra time to study bug reports or relevant code and can avoid unnecessary context switches." (P10)
"The estimates can help prioritize and plan." (Survey)

✓**Help manage customer expectation:** Six interviewees and two survey respondents said that estimating bug fixing time was helpful for managers in that managers could better understand the progress of the whole project and could tell their customers the duration needed to fix their problems.

"To log scope of dev in fixing bug, from that info managers are able to: - Know scope for one sprint −−> deal with customers if scope is expanded - Know performance of dev." (Survey)
"People who aren't as experienced won't be upset that the problem isn't fixed, because often people say like, why don't you fix this this thing? Well, because it's really hard. If you could say it's really hard to fix. That helps people understand why you're not fixing it." (P22)

✓**Save developers' time:** Four interviewees and one survey respondent thought this technique could help save developers' time in estimating bug fixing time. One interviewee and one survey respondent mentioned that developers were bad at estimating the time it took to fix bugs. Another interviewee said that estimating bug fixing time was not easy and would always require an experienced developer to make such an estimation.

"Software developers are very very bad at estimating the time that it takes to develop something or to fix something. Sometimes you said that it will take you a week and it takes five minutes. And sometimes you think that it will take a week and it takes a year. So, something the developers are very, very bad at doing this." (P21)
"If you don't know where it's coming from, you have to spend time setting up a test environment and reproduce it and then figure out what's going on and then modify code for it. It depends...that's why so often you have to have an engineering lead sit there. But then most engineering leads don't want to sit and go through hundred bug reports. It's hard." (P22)

(2) Reasons for unimportant / very unimportant ratings

Some practitioners were negative about the development of such techniques mainly because they did not believe such techniques could work reliably or they did not have a strong need for such techniques.

✗**Questionable reliability:** Six interviewees and fourteen survey respondents thought it was unlikely that the prediction results could be accurate enough to be of much value. Some of them thought bug fixing time prediction was a complex task that required consideration of many factors, while some said bugs were not something they could schedule to resolve.

"Bug fixing time is too unpredictable, or at least depends on too many complex factors for it to be sufficiently precisely predictable. (An estimate of "between 1 and 2 months" is not very helpful.)" (Survey)
"Completely unimportant - a bug can be very easy to fix, or they can be devilishly difficult - and even human estimates are often woefully off, so I don't treat bugs as something you can schedule for resolution." (Survey)

✗**Unnecessary:** Three interviewees and three survey respondents mentioned that there was no need for such a prediction. Three interviewees explained that they had a default fixing time for each kind of bugs in their company. One survey respondent who worked on OSS projects

said he/she just did not care about the bug fixing time. Another survey respondent commented he/she could do this manually.

"In our company, every kind of bugs have a default fixing time. For example, the majority of bugs should be fixed within 1-2 days. Online bugs should be fixed immediately etc. Predicting bug fixing time does not make any sense in our case." (P2)
"In FLOSS, I pretty much don't care. It gets done when it gets done." (Survey)
"I can/need to do this manually." (Survey)

(3) Improvements and Expectations

In terms of features of an effective prediction model, several interviewees provided their opinions as follows:
Consider potential impact on customers: Six interviewees mentioned that the potential impact of a bug on customers especially those best paying ones was one of the most important factors that determined how soon the bug should be fixed.

"How much would a bug affect our customers largely determines how long we should take to fix the bug. If a bug affected our best paying users, e.g. those vip game players, then the bug must be solved as soon as possible." (P23)

Consider fixers' expertise and schedule: Three interviewees suggested including the potential fixers' expertise and their current schedule into the model.

"Different developers will spend totally different time in fixing the same bug. How can you precisely predict a bug's fixing time without knowing who is going to fix it, whether he/she is concentrated on fixing bugs or trying to fix bugs while in meetings...The results will be totally different." (P5)

Consider code complexity and cost of testing: Three interviewees thought that the complexity of buggy code and the cost to test the fixing points were important considerations in predicting bug fixing time.

"When you try to predict a bug's fixing time, you also need to take the code complexity of potential buggy points into consideration. Besides, some bug fixings are very hard to test, which means you also need to consider how much efforts developers make to test their fixes during bug fixing." (P20)

Work on higher granularity level: One interviewee suggested not to provide exact time prediction but to provide a relatively higher-granularity-level prediction, e.g. one day or one week instead of several hours..

"Rather than predict exactly how much time (e.g. several hours) it would take to fix the bug, I think it is more practical to predict, e.g. whether you can fix it in a day or in a week." (P25)

3.3.9 Bug Severity / Priority Prediction (B9)

(1) Reasons for important / very important ratings

Two major reasons were identified during our thematic analysis, i.e. helping people better allocate effort/time and save developers' time in determining the severity/priority of the bugs.
✓**Improve resource allocation:** Nine interviewees and eleven survey respondents commented that identifying bug priority/severity was good for effort/time allocation. This technique could help them focus their efforts on real important things.

"Not all bugs have the same severity/priority. We should always fix high priority/severity bugs to avoid potential bad impact on our business and customers' experience." (P5)
"Especially in open source projects the committers work on parts they are interested in. The interests might not match with the importance of the bugs. Bug reporters often do not change the default priority. Perhaps it is difficult for them to decide how important the bug is. Often estimation of importance is also difficult for committers. This might help that committers prefer working on the most important bugs." (Survey)

✓**Save developers' time:** Four interviewees and two survey respondents mentioned that automated bug severity/priority detection could save developers' time spent in identifying the severity/priority of a bug and help them avoid mistakes in some cases.

"If you let the users set the priority, then every user will set the highest priority for their bugs. This is really not useful, so we always need to find the severity/priority by ourselves. If the tool can work well, then it can save us a lot of time." (P25)
"We do this mostly by manual, but we may make mistakes and this would place the whole project at risk. If some tools can help us to do this effectively, it will do a great help." (Survey)

(2) Reasons for unimportant / very unimportant ratings

We identified two major reasons for unimportant/very unimportant ratings as follows:
✗**Questionable reliability:** Six interviewees and two survey respondents doubted the feasibility of predicting bug severity/priority based just on the textual content of bug reports. They explained that bug severity/priority prediction was also determined by many other factors, such as whether the bug would cause data breach, whether it would affect many users etc.

"Is it possible to precisely predict the severity or priority of a bug just based on the content of a bug report? I do not believe that. For us, we also considered many other things, such as its potential impact on users, or whether it caused data breach." (P15)

✗**Unnecessary:** Three interviewees and five survey respondents thought it was unimportant to develop such techniques mainly because they could do it manually or knowing a bug's severity/priority would not help them too much.

- **Manual work is sufficient.** Three interviewees and two survey respondents said they did not need such techniques because they could do this manually

based on some simple heuristics.

"This is usually quite easy to do based on the conditions under which the bug was reported and some simple heuristics."(Survey)
"Actually in our company, we have detailed heuristic rules describing how to determine a bug's severity/priority." (P8)

- **No practical value.** Two survey respondents thought it was not important to identify a bug's severity/priority. One said the "severity" and "priority" were too coarse. The other one thought prioritization was just a second metric that helped only to sort but not address bugs.

"Prioritization is just a secondary metric that only helps sort, not address bugs. In practice, in a priority ordered list, you'll skip over some of the ones ranked at the top simply because you're not able to solve them yet, even though they are higher priority. So a strictly priority driven view would be prohibitive." (Survey)
"These tools are not useful because "severity" and "priority" are not useful, in my experience. They are too coarse. A bug can render a program unusable for one group of users but be totally irrelevant to many others (for instance, "crash on startup on Windows XP" is critical to those who use Windows XP, but irrelevant to everyone else). Similarly, a bug might be must-fix for release 1.1 but already moot in 2.0 due to unrelated changes – and both releases are being developed in parallel, so release managers need to track both." (Survey)

(3) Improvements and Expectations

Explain prediction result: Eight interviewees and three survey respondents thought that simply providing prediction results was insufficient, extra evidence should be provided to convince developers. Such evidence could be, e.g. the number and type of users affected by the bug, whether the bug is exploitable for malicious purposes, the current schedule/strategy/progress of a project etc.

"When you predict something is bad, you need to explain why. For example, does this bug block system's execution? How many users might be affected by it, how much time should we take to fix it, what is the potential economic loss etc. Sometimes, a VIP customer's phone call would make a low severity/priority bug rank high in our to-be-fixed bug list." (P15)
"It would be useful if the tool can tell which releases of a program are affected by a bug and the likelihood of a bug being exploitable for malicious purposes." (Survey)
"A project's strategy and progress would also dynamically affect a bug's severity/priority. I think this should also be considered." (P23)

Correctly classify high priority / severity bugs: Two interviewees mentioned that an ideal bug severity/priority tool should at least have a high accuracy when predicting such bugs and that it should not miss detecting them. Otherwise it would prove to be impractical.

"I think a very important aspect of a practical bug severity/priority prediction tool should be: if a bug has a high severity/priority, you should not miss it." (P23)

3.3.10 Re-opened Bug Prediction (B10)

(1) Reasons for important / very important ratings

Two reasons were revealed during thematic analysis, i.e. helping avoid incomplete bug fixes and helping track a bug's status.

✓**Avoid incomplete bug fixes:** Seven interviewees and three survey respondents mentioned that re-opened bug prediction could help avoid incomplete bug fixes which were thought as complete. As explained by several interviewees, developers would pay more attention to the fixing points when being told that the bug report might be re-opened. Besides, two interviewees said that testers would also be more careful to test the changing point if they knew that the fixes might cause other problems and be re-opened.

"If they can do, then it to some extent can help developers realize, oh, this problem I may did not imagine I made fixes. Then I will pay more attention." (P20)
"Re-opened bugs especially those occurred online would have a very bad impact on our service and business. All of us don't want to see these bugs occurred again. If some kind of tools can tell us certain bugs (especially those with high priority) are very likely to be reopened, our testers would pay much more attention to testing the bug fixing code." (P8)

✓**Help users track their bugs:** One interviewee and two survey respondents thought this kind of technique could help users know the status and progress of the bug they reported, whether it was partly fixed, completely fixed or when it would be fixed at all. One survey respondent mentioned that this could further reduce developers' burden in handling extra bug reports caused by users' unawareness of the current state of the bugs.

"That's the interesting thing with a history of a bug, trying to get a sense of what's going on. But often it's the case if you think you fixed it, it turns out you didn't fix their exact problems ... Often it'll be that you fixed it, but not entirely. The user reported this, we did part of it, we marked it fixed, but here's the rest..." (P22)
"It helps end user or tester to look old report and file a better report or no report at all. This way developer can save time to review lot of bug reports." (Survey)

(2) Reasons for unimportant / very unimportant ratings

Practitioners considered this technique unimportant mainly because they did not have a need for such a technique or they doubted its reliability.

✗**Unnecessary:** Five interviewees and nineteen survey respondents did not feel the need for such kind of techniques due to various reasons. Some said re-opened bugs were very rare; some said they did not care whether a bug was re-opened or not; some thought such predictions would be saying to developers that their bugfixes were bad; some thought they could do nothing even if they knew

the bug would be re-opened; some thought this technique would inject additional uncertainty into the maintenance process.

[Uncommon] *"We seldom met re-opened bugs. Those re-opened bugs we met before are either previous bugs reintroduced by developers when implementing new features or are performance bugs which are very hard to fix completely. (P5)"*
"Re-opened bugs? We never encountered such bugs." (P3)
[Irrelevant] *"If it's re-opened, it's re-opened, it's just another (particularly annoying, but still just another) bug." (Survey)*
[May offend developers] *"It sounds like you were being told your bugfix is bad." (P25)*
[Not actionable] *"This doesn't appear to be a particularly useful data point. I don't see what you can do if you can preempt a bug being reopened." (Survey)*
[Introduce additional uncertainty] *"The only real purpose of something like this seems to be to inject additional uncertainty into the process and a competent maintainer doesn't need that. In a not-FLOSS context, this just strikes me as a big brother-y way for management to further misunderstand the process of making software. I worry it'd be a bad metric, one that furnishes a numerical excuse for imposing bad process changes that end up doing more harm than good." (Survey)*

✗**Questionable reliability:** Two interviewees and five survey respondents doubted whether it was really possible to predict whether a bug would be re-opened or not: some thought only by testing can we know this; some did not believe such a tool could have more insights than themselves; some thought this kind of technique could not help in those reopened bugs that were caused by an inadequate or breaking fix.

[Unlikely to work unless testing is involved]*"It sounds like science fiction that the software could predict whether or not the bug will be reopened. I don't see how unless it's able to run tests." (P18)*
[Unlikely to provide additional insight]*"I don't know what basis such a tool would operate on, but unless it has a lot more insight into the code than I do, it won't be telling me much." (Survey)*
[Unlikely to work due to deep analysis needed]*"Reopening sometimes happens because the original report was unclear and the fix did not address the issue as perceived by the reporter. You could possibly alleviate this by detecting weasel words, bad grammar or just lack of detail in the report. I believe (but I have no statistics to back this up) that the majority of bug reports are reopened because of an objectively inadequate or breaking fix and I don't think algorithms are going to help here." (Survey)*

(3) Improvements and Expectations
Explain prediction result: Five interviewees mentioned that simple prediction was not enough and that more evidences should be provided to persuade developers to believe the prediction results. These evidences could include, for example, extra testing failures, buggy patterns within code, fixing the wrong problems, better ways to fix the bug, extra cost to completely solve the problem, or potential impact on business if the bug is not specially treated etc.

"I think it would be good if you can tell us how much extra effort we should take to completely fix the re-opened bug." (P10)
"If you found my bugfixes match with some buggy patterns, then I may accept your prediction results. If you can further tell us the potential business impact if we do not pay attention to your prediction, then it will be better." (P23)
"You should tell me the reasons why my bug would be reopened. Did you find any bugs within my bugfix through extra tests? Or you know the better way to fix the problem? Or you can tell that I fixed the wrong problems?" (P25)

Incorporate additional relevant information: Four interviewees thought other kinds of information, besides bug reports, should be made use of to improve the performance of re-opened bug prediction tools. Some of them said that initial fixer's expertise (e.g. whether he/she was good at fixing this kind of bugs) and macro development environment (e.g. whether the bug was fixed under the pressure of releasing) should also be considered while conducting re-opened bug prediction. Some others said whether developers were willing to fix the bug and whether the bug was complex or not were also indicators of re-opened bug prediction.

"I think the first fixer's expertise is very indicative in determining whether a closed bug will get re-opened. I mean, we tend to make mistakes when we do not have enough expertise in fixing bugs. Meanwhile, fixing bugs under great pressure such as before releasing is also error-prone. All of these should be considered." (P15)
"Sometimes you also need to consider whether the fixer is willing to fix the bug. If he/she is reluctant to resolve the bug, then he/she may not carefully fix it." (P23)
"The complexity of the problem would also affect the likelihood to re-open a bug." (P13)

4 Discussion

In this part, we first present some general insights across categories. Then we provide detailed insights into specific techniques. Lastly, we discuss the threats to the validity of our study.

4.1 General insights across categories

Practitioners view bug report management techniques as important. From the rating results (Table 4), we found that on the whole, automated bug report management techniques were considered as important by the majority of practitioners: 6 categories received `70%–82.6%` important/very important ratings; 4 categories received `46.1%–69%` important/very important ratings (these 4 lower-rated categories received a relatively larger ratio of neutral ratings, ranging from 25.6% to 32.2%).

By analyzing the rationale provided by interviewees/survey respondents in Section 3.3, we found that overall practitioners with different roles perceived various benefits in automated bug report management techniques. Some often-mentioned benefits were: speeding up bug fixing, saving manual work in managing bug reports,

helping developers/teams make better schedules etc. These potential benefits not only supported the value of past research in automated bug report management techniques, but also highlighted the importance of advancing such techniques in the future.

Adoption depends on how convinced developers are on the reliability of recommendations made by bug report management techniques. Although the general perspective was positive, some of our interviewees and survey respondents were negative towards the development of some automated bug report management techniques. By analyzing the negative comments in Section 3.3, we found that "questionable reliability" was a common theme that cut across almost all categories of bug report management techniques. This indicates a necessity of further work to make those techniques more reliable/convincing. Some points (towards individual techniques) that were suggested by survey/interview participants in Section 3.3 (e.g. providing explanations for recommendation results) may provide some hints in this direction.

More than one source of data needs to be considered to improve bug report management techniques. In the Improvements and Expectations part of Section 3.3, some interviewees/survey respondents expressed their opinions on how to make bug report management techniques more reliable/convincing. From those comments, we identified one commonly-mentioned direction that researchers/tool builders may take on: making use of other kinds of information. It would be useful for researchers and relevant tool builders to find out the exact information practitioners usually used when doing their day-to-day bug report management tasks in practice. For example, we found that the effects on paying customers was highlighted as a vital factor in determining how to handle a bug – and this has not been considered in prior work on bug severity/priority prediction (B9).

Software engineering practice is diverse; it is essential to better characterize context and situations where a particular bug report management technique is especially needed. A number of opposing comments were given by practitioners. Take bug categorization as an example, some practitioners commented, "I can do this manually" and "it's easily done manually in a good issue tracker", while others commented, "I've become a bottleneck in categorising bug reports". The same situation is observed for bug-commit linking techniques. These examples may indicate that we need to perform an in-depth analysis on characterizing situations where some specific techniques are needed. It would be worthwhile if some effort can be made to investigate what factors (such as project size/type, developer background/skill/role etc.) can affect the adoption of specific techniques. Gaining a better understanding of these would help us develop truly useful techniques for those practitioners who are in need of help.

Difficult cases deserve more attention. As often-mentioned by our interviewees/survey respondents in Section 3.3, there were several specific kinds of bugs bothering practitioners a lot during their bug report management practice. These bugs included performance bugs, bugs that hid in imported third-party components, environment-related bugs caused by hardware crash or bad network, duplicate bugs with different symptoms but with the same root causes, and those that involved several components/products etc. Considering that most existing studies mainly focus on building general purpose models for managing bug reports, it would be very valuable if more effort could be spared to particularly address these specific kinds of bugs to better mitigate practitioner's real gripes and issues.

It is worthwhile to integrate research prototypes to commonly used tools. As mentioned above, all categories of automated bug report management techniques are perceived as important by most practitioners. It would be very valuable to effectively integrate research prototypes that implement bug report management techniques into existing bug tracking systems such as Bugzilla and JIRA. Both Bugzilla and JIRA allow developers to create add-ons (aka. plug-ins)[9][10] and existing research prototypes can initially be introduced as add-ons to the default Bugzilla and Jira instances. Thung et al. have endeavored to do this for duplicate bug report detection and bug localization [74], [75]. However, the plug-ins are only for Bugzilla and they are not actively maintained and advertised. Also, the techniques implemented in their prototypes are dated (i.e. [60], [91]) and more advanced techniques have been proposed in the literature (e.g. [29], [83], [44]).

The creation of such add-ons can help improve the process of bug management, from bug reporting to bug fixing: when users submit a bug report, the system can automatically indicate whether this bug has been reported, or provide essential information to help him/her complete a bug report; after the bug is reported, the system can automatically tell triagers whether this bug is severe/valid or not, and further indicate who is suitable to handle this bug; when developers are trying to fix the bug, this system can help to locate potential buggy code and link the bug report with related commits after it is fixed.

4.2 Insights into specific categories

Our survey respondents and interviewees provided comments to specific bug report management techniques. We can draw some insights from their comments. We highlight them below:

4.2.1 Bug Localization (B1)

More efforts should be made to characterize capability of existing techniques. Several interviewees doubted that bug localization techniques could locate difficult bugs. Such perspective was also reported by [41]. Thus, it would be valuable to conduct empirical studies to investigate characteristics of bugs that are successfully located by existing techniques. This can highlight strengths and weaknesses of current state-of-the-art and help future studies to focus on types of bugs that are not handled well yet.

9. https://www.bugzilla.org/docs/4.4/en/html/extensions.html
10. https://developer.atlassian.com/jiradev/jira-platform/building-jira-add-ons

More pieces of relevant information should be incorporated to improve prediction performance. Many interviewees said that they often manually investigated attached screenshots, stack traces and logs to locate bugs. To the best of our knowledge, most studies only considered bug report text and project source code [91], [61], [54]. A few recent studies considered additional sources of information such as stack traces [52], [83] and logs [44], [42]. However, none had comprehensively used all pieces of information that developers often investigate to locate bugs. For example, none had made use of screenshots to better locate bugs. Future studies may want to use advanced image processing techniques to analyze these screenshots to better locate bugs.

Specialized techniques are needed for important families of bugs. As highlighted in Section 3.3, there were several specific types of bugs that bothered developers much; they included performance issues, environment-related bugs (e.g. those caused by bad network or hardware failure), memory leak etc. Existing bug localization techniques are often designed for generic bugs. It would be worthwhile to design specialized bug localization techniques that are targeted to locate several high-value families of bugs much more accurately by considering unique characteristics of those bugs.

4.2.2 Bug Assignment (B2)

Developers' availability need to be considered. Several interviewees mentioned that the consideration of bug type (e.g. memory leak, null pointer etc.), developers' expertise (e.g. what kinds of bugs they are good at fixing) and developers' availability (e.g. whether they are on a vacation or are participating in many other projects) was essential for developing a good bug assignment tool. It would be interesting for future research to utilize *all* such kinds of information to improve automated bug assignment techniques. Existing research (e.g. [32], [4], [85]) had mainly considered solving the problem of inferring developers' expertise and expertise needed to fix a bug report, and matching those. However, none of them had taken developers' availability into consideration. Ignoring this may cause delay in the handling of bug reports and make such techniques less practical.

Pairwise bug assignment is worth a try. One interviewee highlighted the value of assigning a bug to a pair of developers (i.e. an experienced developer and a novice one); it may be worthwhile to design suitable automated bug assignment tools that can assign bug reports to such pairs of developers.

4.2.3 Bug Categorization (B3)

Screenshots should be fully made use of. Some interviewees mentioned that screenshots within a bug report were very useful in categorizing bugs. To date, there is no study that tried to make use of screenshots to classify bug reports. It would be interesting to see future studies that incorporate image processing techniques to improve bug categorization tools further by making use of these screenshots.

It would be very helpful to automatically identify non-reproducible bug reports. We found that practitioners paid special attention to reproducible bug reports. Some researchers had conducted studies on the reproducibility of bug reports: Erfani et al. explored the characteristics of non-reproducible bug reports by manually checking 32,000 non-reproducible reports [21]. Chaparro et al. tried to detect the missing information (e.g. steps to reproduce) in bug descriptions [14]. However, no existing work have developed a solution that can automatically identify non-reproducible reports. Novel solutions leveraging natural language processing and search-based algorithms can possibly be designed to automatically replay bug report descriptions to see if the same issue can be reproduced. Other possibilities include the design of text classification solutions built on top of a set of well engineered features that can differentiate reproducible from non-reproducible reports.

More work on root-cause-based bug categorization is needed. Some interviewees hoped researchers can help them to classify bugs based on their root causes. There had been several studies trying to address this problem with the help of machine learning techniques [77], [76], [72]. Thung et al. proposed a technique that can classify bug reports into three categories, i.e. control and data flow, structural and non-functional [77], [76]. Tan et al. tried to automatically identify memory and semantic bug reports in Mozilla [72]. However, the root cause categories they considered are coarse grained. The extent to which such coarse grained root cause categories help developers is still unclear. Additionally, existing work only evaluated the proposed solutions on a few projects and their accuracy need to be improved. Thus, there is much room for future research in this topic.

4.2.4 Duplicate/Similar Bug Detection (B4)

It is interesting to study the potential of duplicate bug reports in facilitating bug localization and repair. We noticed that some respondents commented that detecting duplicate/similar bug reports could provide hints for bug fixing. This, to some extent, coincided with the core idea of an existing study [8] that utilized duplicate bug reports to facilitate bug assignment. It would be interesting to investigate whether it is possible to make use of duplicate bug reports to facilitate automated bug localization and repair. For example, one can consider identifying and merging pertinent pieces of information within duplicate bug reports, and use them as hints to identify buggy code and generate repair candidates. Past studies on bug localization (e.g. [54], [44], [83]) and automated repair (e.g. [45]) had not considered this opportunity.

It would be valuable to find ways to identify duplicate bug reports with the same root cause but different symptoms. Several interviewees told us that duplicate bug reports with the same root cause but different symptoms could not be easily identified by developers. These bug reports often required developers to perform deep investigation into the source code. Existing studies mainly exploited explicit or latent textual similarity between bug reports to detect duplicate reports [71], [55]. A few tried to utilize structural information or execution traces within bug reports to

facilitate duplicate bug detection [84], [70]. However, it is still a question mark whether these techniques can identify those duplicate bug reports with same root cause but different symptoms -- textual descriptions of such reports may differ much from one another. Future studies are needed to investigate applicability of existing techniques to such hard cases and to design more suitable solutions if the existing ones fail.

More empirical studies are needed to explore characteristics of duplicate bug reports. We were advised by interviewees to investigate what kind of bugs tended to be duplicated, and the characteristics of duplicate bug reports from different domains. This request, to some extent, agreed with [7] who reported that practitioners hoped data scientists could help them to answer which bugs they commonly made. It would be valuable if researchers spare some effort to study this problem to help developers better understand their products and mistakes and identify problems that may affect many users.

4.2.5 Bug Completion/Refinement(B5)

Privacy is an important consideration. One survey respondent mentioned that "having a high quality automatically reported bug report that adequately guards privacy and what caused it would be very useful!". This points to a new direction for existing bug report completion studies that designed techniques to automatically generate bug reports, e.g. [50], [51], to consider the privacy issue. Ideas from existing studies that tried to preserve privacy of low level testing and debugging data [11], [10], [20] can potentially be adapted to manage the privacy issue.

4.2.6 Bug-Commit Linking (B6)

It would be valuable to explore the possibility of using bug-commit links to create better bug report summaries. Among the positive comments, one commented on the value of moving between bug reports and bug-introducing commits. This comment highlighted the possibility of using bug-commit links to create better bug report summaries that incorporate information from bug introducing commits. This summary will present a more complete story that can potentially better help developers avoid, detect, and fix similar bugs happening in the future.

4.2.7 Bug Summarization/Visualization (B7)

Extra effort is needed to explore the weaknesses / strengths of existing techniques. When being asked what makes a good summary, some interviewees said that a good summarized bug report should contain several items, such as a brief description of the bug, reproducible steps, potential impact on users/business and some short key exceptional logs if possible. To the best of our knowledge, existing studies mainly focused on generating a summary by extracting sentences from the description and comments of a bug report [59], [47]. Whether such sentences contain the above-mentioned items (e.g. steps to reproduce) that are desired by practitioners has no been explored yet. It would be worthwhile to conduct further investigation into this problem. Such investigation can highlight strengths and weaknesses of current state-of-the-art techniques and help future studies focus on extracting important items that are often missed by existing bug summarization techniques.

4.2.8 Bug Fixing Time Prediction (B8)

Additional information can be used to improve prediction performance. Practitioners pointed out that besides the textual content of bug reports, other factors such as developer expertise and their schedule, bug severity/priority, potential impact on business/customers, complexity of source code changes and cost involved to test the fixed code were also important to be considered in predicting bug fixing time. Several kinds of information mentioned above have been exploited by existing studies; they are developer expertise [28], [34], bug severity/priority [58], [88] and complexity of source code changes [12], [87]. However, other information such as developer schedule and testing cost have not been utilized yet. Future studies can be performed to investigate possibilities to include such information to improve automated bug fixing time prediction tools.

4.2.9 Bug Severity/Priority Prediction (B9)

Comprehensive explanations should accompany prediction results. Some practitioners said that it would help if such technique provided some explanations of its prediction results. To the best of our knowledge, existing studies mainly focused on applying machine learning or information retrieval methods to bug reports to build bug severity/priority prediction models [43], [79], [80]. None of them ever reported comprehensive explanations to the predicted results. It would be helpful if future research investigates ways to present extra evidence (that developers can appreciate) to backup prediction results.

Domain knowledge can be integrated into prediction models to boost their performance. Some interviewees highlighted that customers' interests and potential economic lost caused by bugs would greatly affect the assessment of bugs' severity/priority. They said that bugs reported by vip customers always had a higher priority; and bugs that might cause large economic lost or affect many users ranked higher and should be fixed as soon as possible. It would be valuable if such domain knowledge from individual project can be mined and integrated into the prediction model. To achieve this, as indicated in [7], enhancing the collaborations between academic and industry might be a good solution.

4.2.10 Re-opened Bug Prediction (B10)

Further investigation into the potential impact of re-opened bugs and the extra effort required to completely fix them is needed. Several practitioners rated this technique as unimportant mainly because it did not provide more informative information on the prediction results. Why is a bug report predicted as likely to be re-opened? What is the impact of reopening a particular bug report on development process and company's business? How much extra effort is needed to resolve a bug report that is likely to be reopened? To gain some insights into the first question, Zimmermann et al. comprehensively investigated various reasons for re-opened bugs [92]. Shihab et al. performed top node analysis

to explore the factors that were most indicative for a re-opend bug [63]. However, little work has been done to study the potential impact of re-opened bugs and the extra cost to completely fix them. It would be appreciated if such kinds of information is provided to help developers better assess and take suitable actions based on recommendations produced by such re-opened bug prediction tool.

4.3 Threats to Validity

Literature Review. In this paper, we did a literature review to summarize existing studies on automated bug report management techniques. It is possible that we miss some relevant studies. To mitigate this threat, we have taken the following steps: (1) we have considered 17 venues that are major conferences or impactful software engineering journals where papers on bug report management are likely to be published; the conferences and journals were also used as sources for prior literature review studies in software engineering [41], [90], [66], and (2) we chose 2006 (the year when the research area of bug report management took off) as the starting year of papers considered in our literature review. We believe that these steps help to provide a degree of assurance on the completeness of our literature review in covering representative techniques on automated bug report management.

During the process of categorizing existing techniques, we adopted several "checks and balances" to avoid potential subjective bias from a single person. Specifically, two researchers worked together to finish the task. Each step was cross-verified by the two persons and the Kappa value of 0.93 indicates a high agreement on the resulted 10 categories of automated bug report management techniques.

Survey. In this study, we mainly used Likert scale (which was widely used in various research studies across multiple domains, e.g. [7], [46], [41]) to collect practitioners' rating results of automated bug report management techniques. It is possible that some respondents may have provided dishonest answers (e.g. saying what we want to hear or saying what they want us to hear) for various reasons. To help reduce this response bias, we (1) allowed our survey respondents to be anonymous; participants are untraceable if they do not leave email addresses; they can also leave new/anonymous email addresses and (2) guaranteed that no personal information would be disseminated in our paper; we explicitly specified this in our survey invitation emails. As found by Ong et al. [56], anonymity and confidentiality helped in obtaining candid answers from survey respondents. Besides, the majority of our respondents were not our personal acquaintances, this also increased the likelihood of acquiring objective responses.

Some respondents may not understand all the automated bug report management techniques studied in this paper; to alleviate this threat, we provided an additional option (i.e. "I don't understand/prefer not to answer") for each question in our survey, and we found that the option was chosen 29 times (0.89% of all 3,270 ratings). Still, we need to acknowledge that it is possible that some participants overestimate their understanding about the techniques studied in this work. Besides, following the advice in [40] (i.e. using the proper language medium for intended respondents), we translated our survey into Chinese to make sure that respondents from China could understand our survey questions well. We designed our survey in English and Chinese since English is a lingua franca and Chinese is the most spoken language in the world.

Since the survey participants are self-selected, the results/findings might be biased towards those who are more likely/willing to fill the questionnaire (e.g. practitioners with extra free time). As explained in [37], "Avoiding the self-selection principle is impossible", any sponsorship or an encouragement might increase practitioner's participation. To improve the generalizability of our findings, we surveyed 327 practitioners, including both industrial professionals in different companies from different countries and OSS contributors. The number of people we surveyed is similar to prior studies [38], [49].

Since software engineering perceptions and practices are diverse, individual opinions of respondents may not be agreeable to or representative of everyone. We mitigated this by surveying 327 respondents from various backgrounds. We reported all sides of the story capturing diverse opinions from various practitioners.

Interview. Following the strategy of prior studies [65], [1], we stopped our interviews when we reached the saturation of themes after interviewing 25 practitioners. The number of interviewees considered in this study is similar with prior studies, e.g. [25], [33]. We need to acknowledge that opinions provided by our interviewees may not be representative of and agreeable to everyone. Still, considering that our interviewees hold various roles and have different levels of expertise, when combined with the survey results, we believe that our study uncovered various valuable insights regarding software engineers' perceptions of automated bug report management.

Thematic Analysis. Due to the exploratory nature of our study, we used thematic analysis as our research method to analyze the survey comments and transcribed interview data, so as to get a general and specific view of practitioner's perception into the importance of automated bug report management techniques. In order to avoid potential bias from individual's analysis, in this study, each step of thematic analysis was independently performed by two researchers and all disagreements were solved through discussions. Note that although we provided the numbers of interviewees/survey respondents that mentioned certain themes in Section 3.3, we cannot declare the strength or pervasiveness of a theme merely by these numbers because we did not ask each participant about each theme separately.

5 Related Work

In the past decade, researchers have conducted some studies to investigate practitioners' perception of various software engineering research work [46], [7], [49]. Lo et al. investigated how software engineering research were relevant to the practitioners in the field [46]. In their study, 512 practitioners working in Microsoft rated the relevance of 571 papers published in ICSE, ESEC/FSE and FSE. They found

that practitioners were generally positive about the studies done by the research community. Besides, they revealed several fundamental reasons why practitioners considered certain research ideas to be unwise, which is crucial for future software engineering research. Similarly, Carver et al. investigated how industry practitioners perceived the relevance of the work published in ESEM [13]. They found that the ESEM research work was valuable and worthwhile for the software practitioners, meanwhile, they also highlighted some topics for which those practitioners would like to see additional research undertaken. In [18], Devanbu et al. shed some light on the relationship between quantitative evidence and practitioner's belief in software engineering settings. They found that practitioners did have strong prior beliefs based on their experience rather than empirical evidence. Furthermore, to help data scientists in software engineering focus on these important tasks that were perceived by the practitioners, Begel et al. conducted a survey among practitioners and summarized 145 questions that practitioners mostly wanted researchers to solve [7]. The above studies aimed to get practitioners' perspective on the general topic of software engineering. Different from them, our study focused on gaining practitioners' perception on a specific topic in software engineering, i.e. automated bug report management techniques.

Besides the studies mentioned above, some other studies have investigated differences between researchers' and practitioners' perceptions on some *specific* software engineering techniques [41], [57]. Kochhar et al. investigated how practitioners perceive existing research work on automated bug localization techniques [41]. They surveyed 386 practitioners and analyzed factors that affected their adoptions of a bug localization technique; they also highlighted some research directions to develop techniques that mattered to practitioners. This study is most related to our study. Unlike this study, we investigated a wider range of bug management techniques rather than focused only on bug localization. In [57], Palomba et al. studied how developers perceived bad code smells detected by the research community. Particularly, they investigated the extent developers considered detected bad code smells as serious design problems. They evaluated 12 kinds of bad code smells and found that some bad code smells were not considered as threats by the developers. Similarly, Bavota et al. investigated how practitioners perceived software coupling [6]. Specifically, they presented some class coupling that was detected by structural, dynamic, semantic, and logical coupling measures and explored to what extent this coupling aligned with developers' perception of coupling. They found that class coupling caught by semantic coupling measure could better estimate practitioners' mental model. Both of these two studies explored practitioners' perception of the value of a specific research work, i.e. bad code smells and class coupling. Unlike these studies, we investigated a different topic, i.e. automated bug report management techniques.

The studies mentioned above mainly focused on investigating how practitioners perceived the value of *existing* software engineering research work. There were other studies that aimed to help researchers better understand the characteristics of software development and developers' behavior [49], [53], [26], [27], [81]. Meyer et al. conducted a survey among practitioners to understand how they perceived and evaluated productivity [49]. They found that practitioners considered themselves as productive when they finished many or big tasks. Treude et al. explored practitioners' expectations on tools that summarized their activities [81]. Specifically, they studied what kinds of information practitioners wanted the tools to collect, how they summarized their activities and what factors might affect their summary. Murphy et al. [53] conducted an interview with 14 interviewees and a survey among 364 respondents to understand how video game development was different from traditional software development. Gousios et al. investigated the challenges of pull-based development from both the integrators' and contributors' views [26], [27]. In our work, we considered an orthogonal research problem unlike the above studies, i.e. how practitioners perceive automated bug report management techniques.

6 Conclusion and Future Work

In this study, we surveyed 327 practitioners from diverse backgrounds on how they perceive the importance of 10 categories of automated bug report management techniques. Specifically, we asked them to rate the categories and provide the rationale of their ratings. To dig deeper into practitioners' perception, we also conducted follow-up interviews with 25 practitioners.

We found that on the whole, the majority of practitioners saw the value of existing automated bug report management studies. They mentioned various benefits of advancing these techniques, including speeding up bug fixing, reducing manual work in managing bug reports, and helping to make a better schedule for developers/teams, etc. However, some practitioners also expressed their doubts on the effectiveness of those automated techniques. As they explained and suggested, it would be necessary for researchers or relevant tool builders to make use of all necessary sources of information, especially those that practitioners often used in practice, to improve techniques' performance. Besides, for some techniques, e.g. bug fixing time prediction and reopened bug prediction, it would be worthwhile to provide extra evidence or explanation to convince practitioners in a tool's recommendations.

Our study also revealed several specific improvements/expectations practitioners desired during their bug report management practice. For example, we highlighted several kinds of bugs that many interviewees deemed as difficult to resolve, such as performance bugs, environment-related bugs (e.g. those caused by bad network or hardware failure), etc. As another example, some interviewees/survey respondents desired some techniques that could help them solve the privacy problems related to bug reports. It would be appreciated if researchers and relevant tool builders focus more efforts on these particular problems.

In the future, we plan to work on some of these opportunities, e.g. integrate existing research techniques to commonly used tools, perform additional investigation into usage scenarios of various bug report management techniques and improve effectiveness of these techniques by

considering additional domain knowledge we glean from this study.

ACKNOWLEDGEMENT

We are grateful for the survey and interview participants who answered our survey questions and provided many insightful comments. We strongly thank the anonymous reviewers for their constructive comments. Zhenyu Chen and Xin Xia are the corresponding authors. This research is partly supported by National Natural Science Foundation of China (Grant No. 61690201, 61373013), the China Scholarship Council Scholarship.

REFERENCES

[1] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of the 40th International Conference on Software Engineering*, pages 499–510, 2018.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370, 2006.

[3] J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In *Proceedings of the 4th International Workshop on Mining Software Repositories*, pages 2–9, 2007.

[4] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology*, 20(3):10:1–10:35, 2011.

[5] R. A. Armstrong. When to use the bonferroni correction. *Ophthalmic and Physiological Optics*, 34(5):502–508, 2014.

[6] G. Bavota, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. An empirical study on the developers' perception of software coupling. In *Proceedings of the 35th International Conference on Software Engineering*, pages 692–701, 2013.

[7] A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, pages 12–23, 2014.

[8] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful ... really? In *Proceedings of the 24th IEEE International Conference on Software Maintenance*, pages 337–345, 2008.

[9] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[10] P. Broadwell, M. Harren, and N. Sastry. Scrash: A system for generating secure crash information. In *Proceedings of the 12th conference on USENIX Security Symposium*, pages 19–30, 2003.

[11] A. Budi, D. Lo, L. Jiang, and Lucia. Kb-anonymity: A model for anonymized behaviour-preserving test and debugging data. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 447–457, 2011.

[12] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta. How long does a bug survive? an empirical study. In *Proceedings of the 18th Working Conference on Reverse Engineering*, pages 191–200, 2011.

[13] J. C. Carver, O. Dieste, N. A. Kraft, D. Lo, and T. Zimmermann. How practitioners perceive the relevance of esem research. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 56:1–56:10, 2016.

[14] O. Chaparro, J. Lu, F. Zampetti, L. Moreno, M. Di Penta, A. Marcus, G. Bavota, and V. Ng. Detecting missing information in bug descriptions. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, pages 396–407, 2017.

[15] N. Cliff. *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[16] J. Coelho and M. T. Valente. Why modern open source projects fail. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, pages 186–196, 2017.

[17] D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, 2011.

[18] P. Devanbu, T. Zimmermann, and C. Bird. Belief & evidence in empirical software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 108–119, 2016.

[19] M. El Mezouar, F. Zhang, and Y. Zou. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering*, 23(3):1704–1742, 2018.

[20] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems*, 32(2):5:1–5:29, 2014.

[21] M. Erfani Joorabchi, M. Mirzaaghaei, and A. Mesbah. Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 62–71, 2014.

[22] R. A. Fisher. On the interpretation of χ 2 from contingency tables, and the calculation of p. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

[23] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378–382, 1971.

[24] J. Fowkes, P. Chanthirasegaran, R. Ranca, M. Allamanis, M. Lapata, and C. Sutton. Autofolding for source code summarization. *IEEE Transactions on Software Engineering*, 43(12):1095–1109, 2017.

[25] T. Fritz, E. M. Huang, G. C. Murphy, and T. Zimmermann. Persuasive technology in the real world: a study of long-term use of activity sensing devices for fitness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 487–496, 2014.

[26] G. Gousios, M.-A. Storey, and A. Bacchelli. Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296, 2016.

[27] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, pages 358–368, 2015.

[28] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 495–504, 2010.

[29] A. Hindle, A. Alipour, and E. Stroulia. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering*, 21(2):368–410, 2016.

[30] A. Hindle, C. Bird, T. Zimmermann, and N. Nagappan. Do topics make sense to managers and developers? *Empirical Software Engineering*, 20(2):479–515, 2015.

[31] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 111–120, 2009.

[32] H. Kagdi and D. Poshyvanyk. Who can help me with this change request? In *Proceedings of the 17th International Conference on Program Comprehension*, pages 273–277, 2009.

[33] J. Kasurinen, J.-P. Strandén, and K. Smolander. What do game developers expect from development and design tools? In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pages 36–41, 2013.

[34] R. Kikas, M. Dumas, and D. Pfahl. Using dynamic and contextual features to predict issue lifetime in github projects. In *Proceedings of the 13th Working Conference on Mining Software Repositories*, pages 291–302, 2016.

[35] D. Kim, Y. Tao, S. Kim, and A. Zeller. Where should we fix this bug? a two-phase recommendation model. *IEEE transactions on software Engineering*, 39(11):1597–1610, 2013.

[36] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, pages 96–107, 2016.

[37] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, (1):1–1, 2017.

[38] M. Kim, T. Zimmermann, and N. Nagappan. An empirical study of refactoringchallenges and benefits at microsoft. *IEEE Transactions on Software Engineering*, 40(7):633–649, 2014.

[39] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[40] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer, 2008.

[41] P. S. Kochhar, X. Xia, D. Lo, and S. Li. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 165–176, 2016.

[42] G. Laghari, A. Murgia, and S. Demeyer. Fine-tuning spectrum based fault localisation with frequent method item sets. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 274–285, 2016.

[43] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. In *Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, pages 249–258, 2011.

[44] T.-D. B. Le, R. J. Oentaryo, and D. Lo. Information retrieval and spectrum based bug localization: better together. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 579–590, 2015.

[45] C. Liu, J. Yang, L. Tan, and M. Hafiz. R2fix: Automatically generating bug fixes from bug reports. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation*, pages 282–291, 2013.

[46] D. Lo, N. Nagappan, and T. Zimmermann. How practitioners perceive the relevance of software engineering research. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 415–425, 2015.

[47] R. Lotufo, Z. Malik, and K. Czarnecki. Modelling the 'hurried' bug report reading process to summarize bug reports. *Empirical Software Engineering*, 20(2):516–548, 2015.

[48] J. H. McDonald. *Handbook of biological statistics*. Sparky House, 2009.

[49] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 19–29, 2014.

[50] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk. Auto-completing bug reports for android applications. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 673–686, 2015.

[51] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *Proceedings of the 9th IEEE International Conference on Software Testing, Verification and Validation*, pages 33–44, 2016.

[52] L. Moreno, J. J. Treadway, A. Marcus, and W. Shen. On the use of stack traces to improve text retrieval-based bug localization. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, pages 151–160, 2014.

[53] E. Murphy-Hill, T. Zimmermann, and N. Nagappan. Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development? In *Proceedings of the 36th International Conference on Software Engineering*, pages 1–11, 2014.

[54] A. T. Nguyen, T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 263–272, 2011.

[55] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79, 2012.

[56] A. D. Ong and D. J. Weiss. The impact of anonymity on responses to sensitive questions. *Journal of Applied Social Psychology*, 30(8):1691–1708, 2000.

[57] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, and A. De Lucia. Do they really smell bad? a study on developers' perception of bad code smells. In *Proceedings of the 30th IEEE international conference on Software maintenance and evolution*, pages 101–110, 2014.

[58] L. D. Panjer. Predicting eclipse bug lifetimes. In *Proceedings of the 4th International Workshop on mining software repositories*, pages 29–32, 2007.

[59] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 505–514, 2010.

[60] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*, pages 499–510, 2007.

[61] R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry. On the effectiveness of information retrieval based bug localization for c programs. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*, pages 161–170, 2014.

[62] A. J. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30(3):507–512, 1974.

[63] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-I. Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042, 2013.

[64] D. Silva, N. Tsantalis, and M. T. Valente. Why we refactor? confessions of github contributors. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 858–870, 2016.

[65] L. Singer, F. Figueira Filho, and M.-A. Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, pages 211–221, 2014.

[66] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering*, 31(9):733–753, 2005.

[67] D. Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[68] M. Staron, W. Meding, and B. Söderqvist. A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. *Information and Software Technology*, 52(10):1069–1079, 2010.

[69] A. Strauss and J. M. Corbin. *Grounded theory in practice*. SAGE, 1997.

[70] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262, 2011.

[71] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 45–54, 2010.

[72] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai. Bug characteristics in open source software. *Empirical Software Engineering*, 19(6):1665–1705, 2014.

[73] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1):1–18, 2017.

[74] F. Thung, P. S. Kochhar, and D. Lo. Dupfinder: integrated tool support for duplicate bug report detection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 871–874, 2014.

[75] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo. Buglocalizer: integrated tool support for bug localization. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 767–770, 2014.

[76] F. Thung, X.-B. D. Le, and D. Lo. Active semi-supervised defect categorization. In *Proceedings of the 23rd International Conference on Program Comprehension*, pages 60–70, 2015.

[77] F. Thung, D. Lo, and L. Jiang. Automatic defect categorization. In *Proceedings of the 19th Working Conference on Reverse Engineering*, pages 205–214, 2012.

[78] F. Thung, D. Lo, L. Jiang, F. Rahman, and P. T. Devanbu. When would this bug get reported? In *Proceedings of the 28th IEEE International Conference on Software Maintenance*, pages 420–429, 2012.

[79] Y. Tian, D. Lo, and C. Sun. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *Proceedings of the 19th Working Conference on Reverse Engineering*, pages 215–224, 2012.

[80] Y. Tian, D. Lo, X. Xia, and C. Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5):1354–1383, 2015.

[81] C. Treude, F. Figueira Filho, and U. Kulesza. Summarizing and measuring development activity. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, pages 625–636, 2015.

[82] J. Wang and H. Zhang. Predicting defect numbers based on defect state transition models. In *Proceedings of the 6th International Symposium on Empirical Software Engineering and Measurement*, pages 191–200, 2012.

[83] S. Wang and D. Lo. Amalgam+: Composing rich information sources for accurate bug localization. *Journal of Software: Evolution and Process*, 28(10):921–942, 2016.

[84] 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 *Proceedings of the 30th International Conference on Software Engineering*, pages 461–470, 2008.

[85] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu. Towards effective bug triage with software data reduction techniques. *IEEE transactions on knowledge and data engineering*, 27(1):264–280, 2015.

[86] J. Xuan, H. Jiang, Z. Ren, and W. Zou. Developer prioritization in bug repositories. In *Proceedings of the 34th International Conference on Software Engineering*, pages 25–35, 2012.

[87] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan. An empirical study on factors impacting bug fixing time. In *Proceedings of the 19th Working Conference on Reverse Engineering*, pages 225–234, 2012.

[88] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *Proceedings of the 35th international conference on software engineering*, pages 1042–1051, 2013.

[89] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo. Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software*, 117(C):166–184, 2016.

[90] T. Zhang, H. Jiang, X. Luo, and A. T. Chan. A literature review of research in bug resolution: Tasks, challenges and future directions. *The Computer Journal*, 59(5):741–773, 2016.

[91] 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 *Proceedings of the 34th International Conference on Software Engineering*, pages 14–24, 2012.

[92] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy. Characterizing and predicting which bugs get reopened. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1074–1083, 2012.

[93] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, 2010.