Information Needs in Bug Reports: Improving Cooperation Between Developers and Users

Silvia Breu* silvia.breu@comlab.ox.ac.uk
Rahul Premraj[‡] rpremraj@cs.vu.nl
Jonathan Sillito[+] sillito@ucalgary.ca
Thomas Zimmermann[+¶] tz@acm.org (contact author)

* Computing Laboratory, University of Oxford, UK
[+] University of Calgary, Canada
[‡] VU University Amsterdam, The Netherlands
[¶] Microsoft Research, Redmond, USA

ABSTRACT
For many software projects, bug tracking systems play a central role in supporting collaboration between the developers and the users of the software. To better understand this collaboration and how tool support can be improved, we have quantitatively and qualitatively analysed the questions asked in a sample of 600 bug reports from the MOZILLA and ECLIPSE projects. We categorised the questions and analysed response rates and times by category and project. Our results show that the role of users goes beyond simply reporting bugs: their active and ongoing participation is important for making progress on the bugs they report. Based on the results, we suggest four ways in which bug tracking systems can be improved.

Author Keywords: Bug Reports, Information Needs, Questions, Response Rate, Response Time, Question Time

ACM Classification Keywords:
D.2.5 [*Software Engineering*]: Testing and Debugging; D.2.7 [*Software Engineering*]: Distribution, Maintenance, and Enhancement

General Terms: Human Factors, Management

1. INTRODUCTION
In open-source projects, bug tracking systems are an important part of how teams (such as the ECLIPSE and MOZILLA teams) interact with their user communities. As a consequence, users can be involved in the bug fixing process: they not only submit the original bug reports but can also participate in discussions of how to fix bugs. Thus they help to make decisions about the future direction of a product. To a large extent, bug tracking systems serve as the medium through which developers and users interact and communicate. However, friction arises when fixing bugs: developers get annoyed and impatient over incomplete bug reports and users are frustrated when their bugs are not immediately fixed [5, 15].

CSCW 2010, February 6–10, 2010, Savannah, Georgia, USA.

In order to better understand such communities and how they collaborate and interact with each other, we analysed 600 bug reports from the ECLIPSE and MOZILLA projects. In particular, we focused on *what kind of questions are asked in bug reports* and their answers. Such questions implicitly describe information needs for bug fixing (Section 3). We then analysed different aspects, such as when are questions asked (*question time*), how often are they answered (*response rate*), and how much time takes it to receive an answer (*response time*). For each aspect, we reveal several patterns that help to guide designing better bug tracking tools.

Earlier work on information needs in software development focused on software maintenance tasks [18, 27] and the day-to-day needs of collocated development teams [16]. In contrast, our study focuses specifically on bug tracking and considers the entire life cycle of bug reports, which involves many different tasks, such as triaging, debugging, fixing, testing, and reviewing code. We also consider users who report bugs in our study. More specifically, we make the following contributions:

1. *Catalogue of frequently asked questions in bug reports.* We identified a catalogue of questions posed by both users and developers in bug reports, consisting of eight categories and 40 sub-categories, derived from 947 questions in 600 bug reports for ECLIPSE and MOZILLA. Most questions are related to debugging and fixing the bug. Many questions also request further information or relate to bug triaging activities (Section 4).
2. *Statistical analysis of question time, response rate and time.* For each question, we collected whether and when it was answered. Questions which discuss corrections are more likely to be answered. In contrast, answers to triaging and resolution questions take longer. In MOZILLA, questions addressed to developers are more likely to be answered than questions addressed to users (Section 5).
3. *Qualitative analysis of bug reports.* We analysed bug reports with a low response rate or repeated assign-reassign events. We found that bug reports are fixed faster when the reporter participates. Reassignment of bugs to other developers was also an indicator for progress (Section 6).
4. *Consequences for bug tracking.* Our study has several implications for bug tracking systems, e.g., to become more community-oriented and explicitly address evolving information needs (Section 7).

times and how. The choice of statistical tests for post-hoc analysis depends on the data. We used *t-tests* for question times and response times because their values are continuous. For response rates we used *Chi-squared tests* because they are based on dichotomous data (responded to or not).

In cases where the *category* factor was shown as significant by ANOVA, the post-hoc analyses were conducted by comparing each of the eight categories to the remaining seven combined as opposed to comparing them pairwisely. The motivation to do so was firstly our interest to know the specific categories for which the dependent variable differed from all others. Secondly, increasing numbers of statistical tests progressively increase the chances of false positive errors (i.e., rejecting the null hypothesis when it is true). This likelihood is reduced by applying Bonferroni correction, which is an adjustment to the threshold p-value by dividing it by the number of tests. Since we conducted eight tests to compare the categories, we lowered the threshold for significance to $p \leq .00625$. In the analysis below, we report whether the specific category is significant with and without Bonferroni correction. It is important to note that the categories no longer significant with Bonferroni correction are still important since they influence the dependent variable, but their effect must be interpreted with caution.

Question Time

Analysing question time helps us understand the information needs of developers at different stages of the bug fixing process. We computed question time as the ratio of the time difference between the comment that contained the question and when the bug was reported to the lifetime of the bug report. The resulting value is normalised and ranges from $[0 - 1]$; for the sake of simplicity, we refer to it as *question time* throughout this remainder of the paper. The two ANOVA models yielded that *category* ($p < .001$), *topic* ($p < .001$) and *project* ($p < .05$) influence question time, but *addressee* has no significant effect.

The ANOVA result for *category* suggests that information needs of developers change during the lifetime of bug reports. This is also supported by Figure 2, which plots the distribution of question times across different categories. To investigate specifically which category of questions were timed differently from the others, we performed the post-hoc analysis using *t-tests*. We found that questions related to *missing information* and *debugging* are asked early on in the lifetime (both $p < .001$), suggesting that soon after the bug is reported, developers focus on gathering all relevant information related to the bug and to narrow candidate fix locations. On the contrary, questions related to *correction*, *status enquiry*, and *resolution* are asked later on in the lifetimes of bug reports (all three $p < .001$). Note that all categories, with an exception of *status enquiry*, were significant even after Bonferroni correction.

A noteworthy observation in Figure 2 is that question times for all categories range across the full lifetimes of bug reports; especially notable so for bug triaging, which is commonly believed to be undertaken soon after the bug has been reported. Instead, it appears to be an on-going activity that does not necessarily halt after the bug report has been assigned to a developer (see Section 6 for examples).

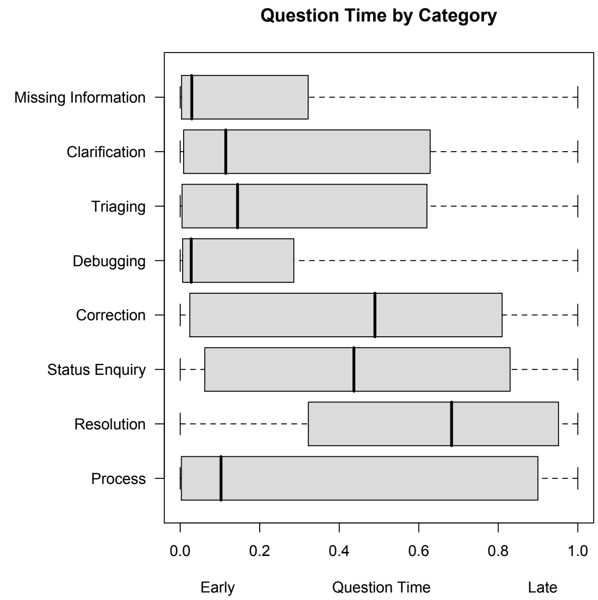

Figure 2. Boxplot of question time by category.

Topic showed also a significant effect on question time in the ANOVA analysis. The post-hoc analysis confirmed this result and showed statistically significant differences in the mean values for the two levels of topic ($p < .001$). Questions related to fixes were asked later in the bugs' lifetimes than questions related to bugs. This indicates that the focus of questions shifts from collecting information related to the bug to how the bug should be fixed.

Project also showed a significant effect in the ANOVA analysis; however, the *t-test* could not confirm any significant effect of project on question times.

Response Rates

As an initial exploration of the challenges developers face in satisfying their information needs, we examined the effect that our independent variables (question category, addressee, topic and project) had on how likely questions were to be responded to. The ANOVA models showed that category of questions has a significant effect on response rates ($p < .001$). ANOVA also showed two interaction terms to be significant, namely *project:addressee* ($p < .001$) and *topic:addressee* ($p < .05$). Although the main factors project, addressee, and topic were also significant, they need no further analysis since they will be covered by the post-hoc analysis on the interaction terms.[6]

[6]Post-hoc analysis on significant interaction terms is conducted by keeping the level of one factor constant and comparing the dependent variable by different levels in the other factor. After repeating this for all levels in the first factor, levels in the second factor are now kept constant to examine differences in the first factor.

Table 1. Response rates for questions by category.

	Replied	Not replied	Total	Response Rate (%)
Missing information	89	54	143	62.24
Clarification	72	42	114	63.16
Triaging	59	35	93	62.77
Debugging	117	59	176	66.48
Correction	189	51	240	78.75
Status Enquiry	56	24	80	70.00
Resolution	38	30	68	55.88
Process	16	9	25	64.00
Total	636	304	940	67.66

The result for the *category* factor indicates that whether a question is likely to receive a response depends on the type of question that has been posed (see also Table 1). In the post-hoc analysis, questions related to *correction* were more likely (response rate of 78.8% vs. 64.1%, $p < .001$) and questions related to *resolution* were less likely to receive responses (56.0% vs. 68.7%; $p < .05$). Note that of these two categories, only *correction* remains statistically significant after Bonferroni correction.

For the interaction term *project:addressee*, the post-hoc analysis revealed that in the MOZILLA project, questions addressed to developers have a significantly higher response rate of 72.2% as compared to questions addressed to users with a response rate of 50.0% ($p < .001$). In addition we found that questions addressed to users in the ECLIPSE project have a higher response rate of 69.3% as compared to those from MOZILLA with 50.0% ($p < .01$). No other significant differences were found for this interaction.

For the interaction term *topic:addressee*, the post-hoc analysis showed that developers are more likely to respond to questions related to fixes than to questions related to bugs (78.2% vs. 66.4%, significant at $p < .01$). No other significant differences were found for this interaction.

Response Times

Another aspect relevant for the analysis of replies is *response time*, i.e., how quickly does the addressee respond to the question. Delayed replies can slow down progress on bug reports, eventually bringing some to a standstill. For example, a developer may wish to clarify the conditions under which she can reproduce the bug. Without a response, she is less likely to make progress on the bug.

Our analysis now focuses on the questions that received responses. Response time for each question was computed as the time difference between the comment in which the question was posed and the first comment in which it was answered (completeness of answers was disregarded). Thereafter, the data was normalised by ranking the response times to meet the data assumptions for modelling using ANOVA. The ANOVA models only showed for *category* a significant influence on response time ($p < .001$). This means that response time depends mostly upon the type of question.

Our post-hoc analysis on the category factor revealed that questions related to *clarification* ($p < .05$) and *process* ($p < .05$) have lower response times, i.e., are replied to quickly. In contrast, responses take longer for questions related to *triaging* ($p < .01$) and *resolution* ($p < .05$). Of these four category types, only *triaging*-related questions were statistically significant after Bonferroni correction.

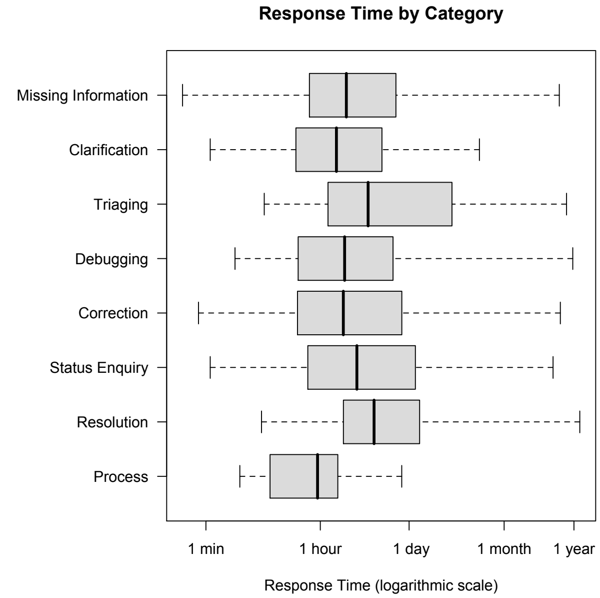

Figure 3. Boxplot of response time by category.

Figure 3 plots the response times by category (note that the plot uses raw response times). There are cases where questions did not receive a response until after a year, for example, it took more than four years to receive a reply for question "Is this still an issue?" in MOZILLA bug 4633. To summarise, although a vast number of questions receive responses quickly, others questions take much longer and and many go unanswered.

Threats to Validity (Statistical Analysis)

As with any empirical study, it is difficult to draw general conclusions because any process depends on a *large number of context variables* [4]. In our case, we analysed 600 bug reports from two large open-source projects, namely ECLIPSE and MOZILLA. We expect that our findings also apply to other projects. The observations made from the statistical analysis are based on 600 randomly sampled bug reports that may not be fully representative of their respective projects. However, the questions identified from the sample cover a vast spectrum of categories that include nearly every aspect of the bug fixing process. We can hence expect that our findings are generalisable and reflective of the projects.

6. QUALITATIVE ANALYSIS

In this section, we discuss bug reports with low response rates and frequent reassignments in more detail. Our analysis of these reports yielded several insights, however, our conclusions should be considered preliminary.

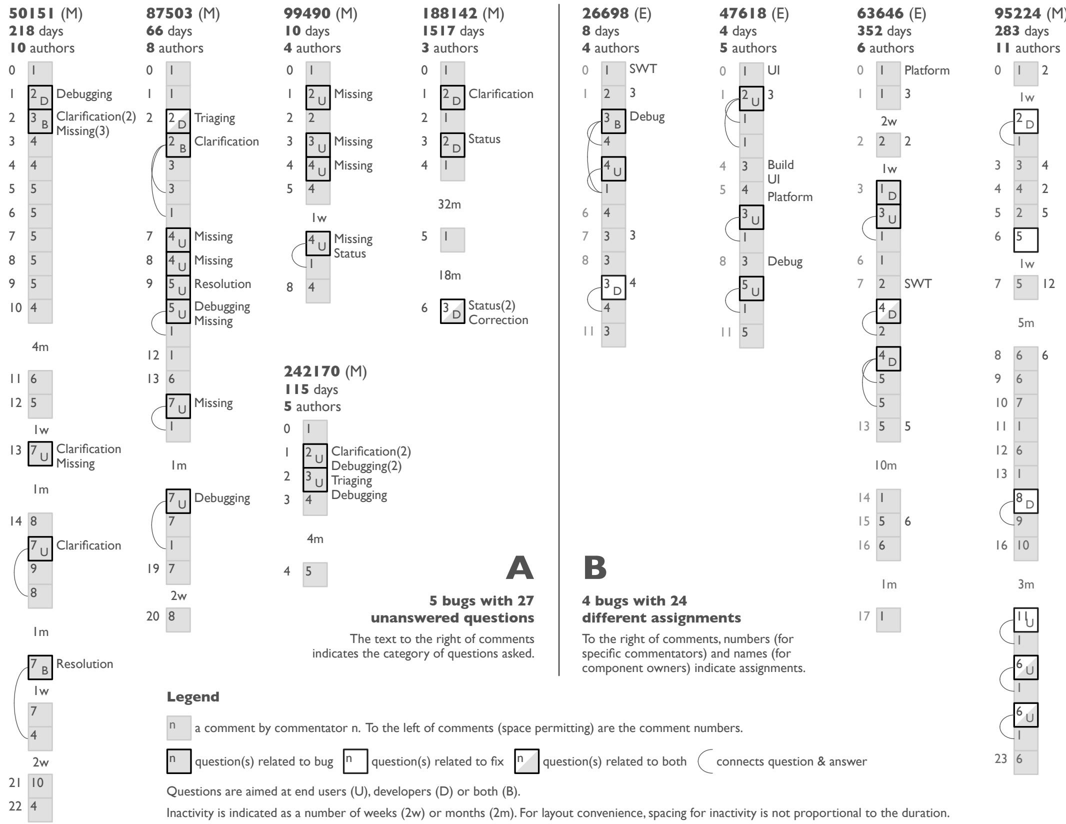

Figure 4. Illustration of discussion in nine bug reports.

Low Response Rate

Section 5 quantifies the challenge developers face in getting their questions answered in a timely fashion. In particular, 32.34% of the questions in our sample were never responded to, and many more required hours or days before an answer was provided. To explore this issue further we have looked more closely at five bug reports that contained many unanswered questions. This way we have been able to carry out a preliminary analysis of 27 unanswered questions in context.

To select the bug reports for this analysis, we considered all reports with at least five questions and selected the five reports with the lowest response rates. The discussions contained in these reports are illustrated in part A of Figure 4. All of these reports come from the MOZILLA project, which we found to have lower response rates in our sample (65.7% versus 71.3% for the ECLIPSE project). As shown in the figure, with the exception of issue 99490, these issues all took a significant amount of time to address and included long stretches of time without discussion.

For the MOZILLA project 50.0% of the questions addressed to users went unanswered, and indeed most of the 27 unanswered questions were addressed to users (20 questions) and are related to the bug rather than the fix (26 questions). For example, during the discussion of bug 242170, developers asked the reporter six questions about the bug (four in comment 1 and two in comment 2) none of which were answered. After several months with no response, one of the developers guessed that the problem had been resolved and marked it as *RESOLVED/WORKSFORME*.

Our qualitative analysis suggests that there is an expectation by developers that users reporting a bug, actively participate in the discussion of the bug if necessary and our statistical analysis showed that questions addressed to users appeared throughout the life cycle of a bug report (see Section 5). A lack of response by users can result in a sense that reporters are not doing their part and that their cooperation is essential for progress to be made. For example, during the discussion of bug 99490, after several questions (asked in comments 1, 3 and 4) had gone unanswered, in comment 6 the developer encouraged the reporter to respond and explained that no work would occur otherwise: *"Reporter, please work with us on this [...] Please comment within the next week or so; otherwise we may have to resolve this worksforme."* Once the reporter responded to some of the questions, which happened in comment 7, the bug was quickly resolved.

The unanswered questions in these five bug reports tended to be about information that is necessary for proper triaging, debugging, and especially to reproducing a bug. A lack of response to such questions is particularly frustrating for developers because without the ability to reproduce a bug, generally no progress can be made and developers tend to mark the bug as *RESOLVED/WORKSFORME* or let it sit idle. For example, during the discussion of bug 50151, a developer asked about the "build number" [comment 13] along with another question clarifying what the bug is about. There was no answer to this question and so no progress was made for one month until a different user added some additional information (in comment 14).

On the other hand, we also observed that some questions were superseded or became irrelevant as the discussion progressed. This is the case for the questions in comments 7, 8, and 9 in the discussion of bug 87503. In comment 7 the commentator asked about what version of MOZILLA the problem occurred in and then clarified in comment 8 that he meant "which build". Commentator 5 suggested that someone checks whether or not the latest build, with a particular patch, still has the problem. The discussion then continued, but only considering the latest build. In cases like these, the lack of a response is not indicative of a problem, though this appears to be the less common case.

Triaging Issues

In Section 5 we noted that triaging-related questions are posed throughout the life cycle of a bug. We also suggested that discussion about *how* to address a bug may influence *who* should address it. Related to this, previous work has identified a bug pattern called the *assign/reassign* cycle and hypothesised that such a pattern may indicate a structural problem in the software or an organisational gap [10].

To further explore assignment and reassignment issues as they relate to questions asked, we have analysed the questions asked in four bug reports; three from the ECLIPSE project and one from the MOZILLA project. To select the bug reports for this analysis, we considered all reports with at least three questions and selected the four with the most reassignments. In this way we have been able to carry out a preliminary analysis of 24 assignment decisions (some to generic component owners, others to individual developers) in context. The discussions contained in these reports are illustrated in part B of Figure 4.

An analysis showed that some questions and the reassignments that followed intend to help move the bug towards resolution in various ways. The most obvious were questions aimed at understanding the issue sufficiently in order to get the assignment correct. For example, during the discussion of bug 47618, the question asked in comment 6 tried to understand under what circumstances the bug occured. The answer in comment 7 led directly to the reassignment in comment 8 to the group responsible for the Debug component.

Similarly, there were questions about who was responsible for a given area and answers to these questions are important for appropriately assigning responsibility for the bug. For example, during the discussion of bug 95224, the bug was assigned to commentator 5 (see comment 5) because it was believed that he was the appropriate module owner. In comment 6 he denied that he was the owner ("when did that happen?") and reassigned the bug to a developer who never participated in the discussion (see comment 7). No further progress was made until commentator 6 claimed the bug five months later (see comment 8).

In a few cases a question was asked and simultaneously the developer who could answer the question (and take the next steps with the bug) was assigned the bug. This happened several times during the discussion of bug 26698. In comment 2, the commentator (who was the assignee at the time) asked a question and reassigned the bug to the group that could answer it ("moving to Debug for comment"). After tying off the ensuing discussion with members of that group, commentator 3 took the bug back. Once he had finished his work on the bug, he asked commentator 4 to verify what he had done and assigned the bug to him (see comment 9). These results suggest that multiple reassignments are not problematic in all cases and can be a natural part of the question and answer process.

7. IMPLICATIONS FOR BUG TRACKING

Previous work discussed various shortcomings of today's bug tracking systems [15] and proposed several improvements such as interactive feedback systems [5] and alternative handling of bug duplicates [6]. Based on the results reported in this paper, we suggest *four new ways* in which bug tracking systems and practices can be improved.

Evolving information needs. From the analysis of question time in Section 5, we learned that the kind of questions and thus the information needs change over a bug's life cycle. In the beginning most questions request missing information or details for debugging (in order to locate the source of the bug). Later questions are mostly focused on the correction of a bug and on status enquiries. A direct consequence is that bug tracking systems should account for such evolving information needs. For example, in the beginning, when more information about a bug is needed, easy ways to provide screenshots, stack traces, or steps to reproduce, are important. Later in a bug's life cycle, such interfaces can be replaced by interfaces that facilitate discussing the correction and tracking the bug's resolution.

Tool support for frequent questions. In our study, some question categories were very frequent, e.g., review requests or status enquiries (see Section 4). Introducing tools that help addressing these questions in a timely and organised manner will streamline bug tracking activities. As an example consider the request for reviewing a suggested fix. If this is clearly assigned by the bug tracking system to a developer, e.g., through a separate work item, a code review is more likely to be completed and is easier to track. Thus, reviewing bug fixes becomes an active rather than a passive part, dependent upon the emergence of a volunteer code reviewer.

Explicit handling and tracking of questions. For the MOZILLA project 50.0% of the questions addressed to users went unanswered. We believe that many users do not understand the jargon used by developers and require explicit requests, like "please work with us on this" (see Section 6). A solution for this problem could be to make the state of the discussion explicit, not just the state of the bug report. For example, developers could mark up crucial questions, which the bug tracking system recognises and puts the bug in a state "answers pending". Making this state explicit sends a clear message to all stake-holders of the bug report. One can take this even further and collect bugs that are stuck because of insufficient information on a special dashboard; they could then be specifically targeted.

Community-based bug tracking. The high number of unanswered questions in MOZILLA could result from users who feel their job is done after initially reporting a bug. This sentiment is heightened by a form design in bug trackers that emphasises reporting of information rather than interaction. To overcome this, bug reporting and tracking should be understood as a social activity within a community, supported by the bug tracking system. For example, it could be more of a project portal, which indicates the assigned developer and her recent activities, the status of a bug, new questions in bug reports, the history of the reporter, including bugs she had reported previously and maybe even reputation of reporters.

Joel Spolsky once noted "I've always felt that if you can make it 10% easier to fill in a bug report, you'll get twice as many bug reports" in his blog [13]. While usability is of importance to us, we focus on improvements after the initial submission of a bug report. Therefore our suggestions are unlikely to increase the number of reports. Instead, we hope that our ongoing research will contribute toward the development of social and interactive bug tracking systems that address the needs of users and developers alike. Our long term aim is to increase the percentage of *fixed* bug reports.

8. CONCLUSIONS

Bug tracking systems are an important part of how teams in open source interact with their user communities. This interaction goes beyond users simply submitting bugs. Many follow-up questions are posed to the reporters of bugs and often, if a reporter does not play an active role in the discussion of the bug, little progress is made. Our results highlight the importance of effectively and efficiently engaging the user community in bug fixing activities, and keeping them up-to-date about the status of a bug. We believe that our results will help to form the design of new bug tracking systems that will aim at eliciting the right information from users and facilitating communication between end users and developers as well as among developers. An integration and active participation of users in bug tracking will result in bugs being fixed faster and more efficiently.

All cards, the categorization, and R scripts to replicate our study are available as a technical report [7].

Acknowledgements.

Many thanks to Rebecca Aiken, Juliane Degner, Christian Lindig, Alan Mycroft, and Andreas Zeller, as well as the anonymous reviewers for valuable feedback on earlier revisions of this paper. Silvia Breu was supported by an Eclipse Innovation Award. Thomas Zimmermann was supported by a start-up grant from the University of Calgary.

9. REFERENCES

1. J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06*, pages 361–370, 2006.
2. J. Aranda and G. Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. In *ICSE '09*, pages 298–308, 2009.
3. I. Barker. What is information architecture? KM Column, available at http://www.steptwo.com.au, April 2005.
4. V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Trans. Software Eng.*, 25(4):456–473, 1999.
5. N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *SIGSOFT '08/FSE-16*, pages 308–318, 2008.
6. N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate bug reports considered harmful... really? In *ICSM '08*, pages 337–345, September 2008.
7. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Appendix to Information Needs in Bug Reports Technical Report 2009-945-24, Dept. of Computer Science. University of Calgary, October 2009.
8. A. Erdem, W. L. Johnson, and S. Marsella. Task oriented software understanding. In *ASE '98*, pages 230–239, 1998.
9. J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382, 1971.
10. C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *CSCW '06*, pages 39–48, 2006.
11. J. D. Herbsleb and E. Kuwana. Preserving knowledge in design projects: what designers need to know. In *CHI '93*, pages 7–14, 1993.
12. N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *DSN '08*, pages 52–61, 2008.
13. J. Spolsky. Joel on Software blog. FogBUGZ. http://www.joelonsoftware.com/news/fog0000000162.html.
14. W. L. Johnson and A. Erdem. Interactive explanation of software systems. *Automated Software Engineering*, 4(1):53–75, 1997.
15. S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *VL/HCC '08*, pages 82–85, 2008.
16. A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *ICSE '07*, pages 344–353, 2007.
17. A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *VL/HCC*, pages 127–134, 2006.
18. A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Software Eng.*, 32(12):971–987, 2006.
19. J. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.
20. Ø. Langsrud. ANOVA for unbalanced data: Use Type II instead of Type III sums of squares. *Statistics and Computing*, 13(2):163–167, 2003.
21. R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.
22. P. C. Rigby, D. M. German, and M.-A. Storey. Open source software peer review practices: a case study of the apache server. In *ICSE '08*, pages 541–550, 2008.
23. G. Ripoche and J. P. Sansonet. Experiences in automating the analysis of linguistic interactions for the study of distributed collectives. *Journal Comput. Supported Coop. Work*, 15(2-3):149–183, 2006
24. P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07*, pages 499–510, 2007.
25. R. J. Sandusky. Information, activity and social order in distributed work: The case of distributed software problem management. *PhD thesis*, University of Illinois at Urbana-Champaign. 2005.
26. R. J. Sandusky and L. Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *GROUP '05*, pages 187–196, 2005
27. J. Sillito, G. C. Murphy, and K. D. Volder. Questions programmers ask during software evolution tasks. In *SIGSOFT '06/FSE-14*, pages 23–34, 2006.
28. X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE '08*, pages 461–470, 2008.