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).

2. RELATED WORK

In this section we compare and contrast our work to previous research in two areas. First, we consider studies that investigate the information needs of developers in various contexts. Information needs of developers in collocated development teams were studied by Ko et al. [16]. The authors observed the daily work of developers and noted the types of information desired—they identified 21 different information types in the collected data. Sillito et al. [27] examined the kinds of questions that programmers ask during change tasks. An extensive collection of 44 different kinds of questions were identified. Johnson and Erdem [14] examined questions posted to Usenet newsgroups and categorised them into three classes: goal-oriented, symptom-oriented, and system-oriented. Erdem et al. [8] analysed the questions further along with questions from a literature survey to develop a model of questions that programmers ask. Herbsleb and Kuwana [11] focused on software designers and investigated the types of questions that get asked during design meetings. We are also interested in information needs of developers, however in the context of developers and users collaborating around a bug report.

Second, we consider our work in the context of research that has focused on bug tracking systems. Ko et al. [17] looked at thousands of bug report titles and identified fields that could be incorporated into new bug report forms. Aranda et al. [2] reported on a field study of coordination activities around bug fixing at Microsoft. They identified common bug fixing coordination patterns and provided implications for coordination in software development. Bettenburg et al. [5] conducted a survey on developers and users from APACHE, ECLIPSE, and MOZILLA to determine which information contents, in their opinions, comprise good quality bug reports. Their CUEZILLA tool leveraged the responses from the survey to measure quality of bug reports in real-time and provide immediate feedback to reporters on enhancing information quality. Just et al. [15] analysed the responses from the same survey to suggest improvements to make bug tracking systems easier to use and facilitate submission of better quality bug reports. Lastly, another study by Bettenburg et al. [6] argued for merging of duplicate bug reports along with the originals to make more unique information about the bug available to developers. In contrast, our work focuses on the interaction between developers and users with the goal of improving tool support for this interaction.

Sandusky used primarily qualitative methods on MOZILLA bug reports to describe phenomena such as question and answer sequences [25]. Sandusky and Gasser focused on the role of negotiation in software problem management and how it affects the organisation of information [26]. Ripoche and Sansonnet analysed speech acts across the entire Mozilla Bugzilla corpus computational linguistics [23].

3. DATA COLLECTION

For our study, we analysed the bug repositories of two large open-source projects, ECLIPSE[1] and MOZILLA[2]. We selected these projects because both of them have many developers and users and a long development history.

Combined, both projects have over 600 000 bug reports, which are too many to be analysed manually. Therefore, we used simple random sampling on all bug reports to select 300 bug reports for each project. We then manually identified questions in the 600 bug reports. Next we grouped the extracted 947 questions into categories using a card sort.

Collecting Questions

We extracted questions from bug reports by carefully reading each sampled bug report. Questions come in different forms, with different intentions, and not all of them describe information needs. Therefore, we used the following selection criteria: a question relevant for our study is *any text* that asks for *information or feedback* that is *related to the bug or fix*. In particular, this means that we excluded from our analysis: requests for action (unless they asked for confirmation), rhetorical questions (no response expected), and questions unrelated to the bug or to fixing it.

Here are a few examples to illustrate what we consider to be a question.

- ✔ *"Which operating system are you using?"* → **asks for information related to a bug**
- ✔ *"Why the tabs instead of putting this on the first page of the wizard?"* → **ask for the reasoning behind a proposed fix**
- ✔ *"Would you mind giving the changes a once-over after I make them?"* → **asks a developer for feedback on changes**

In contrast, here are examples of text that we did not consider to be questions for the purpose of our study.

- ✘ *"There is no mozilla bug here, and were we to cater to the obvious bugs of Delphi, then what happens when the next implementer comes along who also misinterpreted the spec and thought the header was mandatory?"* → **rhetorical question, no information seeking**
- ✘ *"Can you commit it to SUNBIRD_0_3_BRANCH (and the other branches/trunk) before todays (sic) nightly build is produced?"* → **request for action, no answer expected**
- ✘ *"What is a child entry [when defining access rules]?"* → **unrelated to the bug and its fix**[3]

For each identified question, we recorded the following information on an index card:

- the *bug id* of the bug report;
- the *number of the comment* that contains the question (the initial bug description has the number 0);
- the actual *question*; if needed we added the *context* to make questions self-contained;
- whether the question is *addressed* at developers, users, or both (i.e., who is expected to respond);

[1] ECLIPSE is a popular integrated development environment for Java and other programming languages. As of July 6, 2008, the bug database for ECLIPSE contained 238 541 bug reports (=our sample frame), dating back to October 2001. http://www.eclipse.org

[2] MOZILLA is a suite of programs for web browsing and collaboration (such as email client, calendar, and address book). As of July 7, 2008, its bug database contained 435 392 bug reports (=sample frame), dating back to April 1998. http://www.mozilla.org/

[3] The question is not related to a bug or fix because a user asked in a bug report how to use the new ECLIPSE feature "type access rule".

- whether the question is *regarding* the bug, fix, or both;
- and *responses*, if any, as a list of comment numbers (we only considered responses within the same report).

After collecting all questions on index cards, we entered all data into a database and added author and time information (via the bug id and comment numbers). With author information, we can distinguish between different roles such as developer, submitter, and assignee. With time information and the comment numbers for questions and responses, we can compute response times.

Card Sort

To group questions into categories, we conducted a *card sort*. Card sorting is an inexpensive sorting technique that is widely used in information architecture to create mental models and derive taxonomies from input data [3]. In our case it helps to organise the questions into hierarchies to deduce a higher level of abstraction and identify common themes. A card sort involves three phases:

1. In the *preparation* phase, participants of the card sort are selected and the cards are created.
2. In the *execution* phase, cards are sorted into meaningful groups with a descriptive title.
3. In the *analysis* phase, abstract hierarchies are formed in order to deduce general categories and themes.

We applied an *open* card sort, meaning there were no predefined groups, instead the groups emerged and evolved during the sorting process. In contrast, a *closed* card sort has predefined groups and is typically applied when themes are known in advance, which was not the case for our study.

All of the cards were created by the first author of this paper. Throughout our further analysis three researchers (first, third, and fourth author) were involved in iteratively developing categories and assigning cards to categories so as to strengthen the validity of the result. The first author played a special role of ensuring that the context of each question was considered appropriately in the categorization, and creating the initial categories. To ensure the integrity of our categories, the cards were sorted by the first author several times to identify initial themes. Next, all researchers reviewed and agreed on the final set of categories as presented in the next section. We measured inter-rater agreement for the first author's categorization on a simple random sample of 100 cards with a closed card sort and two additional raters (third and fourth author); the Fleiss' Kappa [9] value among the three raters was 0.655, which can be considered a substantial agreement [19].

Threats to Validity (Card Sort)

Bystanders of a communication might misplace its context as they may lack full understanding of its nature and background. This threat applied to us, too, when we identified questions and responses in our sample of bug reports. Open card sorts are also inherently subjective because different themes are likely to emerge from independent card sorts conducted by the same or different people.

4. CATALOGUE OF QUESTIONS

After we finished the card sort, we had 40 groups, which we clustered into eight categories.[4] We now describe each category and provide examples of questions. The numbers in parentheses indicate how many times questions in each group were asked.

Category #1: Missing Information

Often, submitted bug reports are incomplete and miss information relevant to reproduce a bug. In fact, this is one of the most frequent problems that developers face with bug reports [5], and is confirmed by our study: *missing information* is the third-largest category. To better understand a bug before starting to debug, developers request information such as steps to reproduce, build numbers, OS, test cases, examples, program output, and screenshots.

→ *in total 143 questions, or 15.1% of all questions.*

steps to reproduce (16×) "How do I actually reproduce this border problem?", "Can you give a description of when this happens to you?"

environment: build, OS, installed software (51×) "Can you provide a build number?", "Which operating system?", "Do you have flash installed?"

tests and examples (16×) "Could you attach test suite or strip it down to a sufficient example?", "What are you trying? Get some real example of your code here…"

program output: log file, talkback, stack trace (32×) "This bug needs a stacktrace (sic).", "Talkback ID from crash?"

miscellaneous information requests (28×) "Could you provide a screenshot?", "Where do you crash in profile manager?"

Category #2: Clarification

Often developers have all relevant information about a failure, but need to clarify certain aspects, which they did not fully understand. The questions in this category relate to the bug in general, and not to possible corrections, which are part of a separate category (*correction*). Clarification questions can be specific or very general. Often developers are even clueless what problem is reported in a bug (*"don't understand, please explain"*). Some questions come from users who want to know if they have been helpful or whether additional information is needed.

→ *in total 114 questions, or 12.0% of all questions.*

asking something specific (16×) "How do you select a different input method?", "Where's a java-applet on this page?"

do you mean? (24×) "Do you mean in the 'browser' tab?", "Are you saying you cannot download a file?"

don't understand, please explain (28×) "I am not clear on what the remain problem is. Can someone explain it to me?", "I don't understand why you would want to export web app libraries?"

questions about provided data (12×) "Was the original dump a 'copy' on the call stack?", "What do the port numbers refer to?"

user regarding his helpfulness (5×) "[Developer,] Is there anything else I can do to help troubleshoot this problem?", "Were you able to reproduce using the steps [I, the reporter,] described in comment #7?"

miscellaneous clarification questions (29×) "Which part do I need to verify for this bug?", "What should the path have been?"

[4]Out of the 947 questions, seven did not fit any of the identified groups and were ignored for the remaining analysis.

Category #3: Triaging

Bug triage is the process of deciding which bugs should be fixed and assigning them to developers. It includes the decision whether a behaviour is actually incorrect (*"bug or feature?"*). Often bugs are submitted to wrong components or even projects, e.g., to MOZILLA instead of JBoss (*"not our bug?"*). Sometimes two different bugs are described within the same report or a new bug emerges during bug fixing (*"separate bug report?"*). Most questions in this category, however, are about bug duplicates and who should fix a bug, both of which have been extensively covered by research [1, 6, 12, 24, 28].

→ *in total 94 questions, or 9.9% of all questions.*

duplicate or not? (35×) "Has this been investigated previously?", "Dupe of bug 114853?"

who could fix it? (10×) "Who gets the Solaris problem?", "Anybody want to own this bug?"

could you fix it? (14×) "Can you do the fix as suggested?", "Could you take care of this?"

bug or feature? (5×) "Is that intended or should it be fixed?", "Is this normal or a bug?"

shall we fix it? (9×) "Is this little nit worth picking?", "Not sure whether we care enough to try to change this?"

separate bug report? (7×) "Should this be written up as a separate 'enhancement' bug?", "Shall we open a new bug or rename the summary of this one?"

correct component? (5×) "Is dom the correct component?", "Can you verify that this is a reconciler problem?"

not our bug? (9×) "Should I move this bug report to JCore-code assist?", "Is this a JBoss bug that needs reporting?"

Category #4: Debugging

This category contains questions related to the process of debugging. It is the second-largest category, which indicates that debugging is a highly collaborative activity, involving both developers and users alike. Ko et al. made a similar observation about the collaborative nature of debugging [16]. In general, developers ask questions about behaviour and state of a program, and about code pieces. Sometimes they require input from users and ask them to rerun programs with different settings.

→ *in total 176 questions, or 18.6% of all questions.*

questions about the behaviour of a program (65×) "Does this exception happen consistently for you?"

questions about state, setting (18×) "Do you have autobuild on?"

questions that require action and/or rerun (42×) "Can you see what happens with a new profile?"

questions about code (25×) "Could it be because the user name field has "type = input" rather than "type = text"?"

questions that ask why or why not (17×) "Why do all the radio buttons have two gray dots now?"

miscellaneous debugging questions (9×) "Any further comments or ideas?"

Category #5: Correction

This category contains questions that discuss how to correct a bug. It is by far the largest category, which indicates that once developers found the cause of a bug, they discuss solutions and alternatives before they fix the bug. The questions in the first group *suggestion & feedback requests* are asked before any changes are made and discuss possible solutions. The questions in *understanding fixes* relate to an implemented solution, often a patch. The last group are questions related to *code reviews*, where developers ask about or get feedback on their solution. Reviewing code is considered an important quality assurance mechanism in the open source community [22].

→ *in total 240 questions, or 25.3% of all questions.*

suggestion & feedback requests (80×) "Should we just remove the getBuiltIn check StreamHandlerFactory?", "Do we want to change all 'click' to 'choose'?"

understanding fixes (87×) "Maybe I'm not understanding this statement but you could use a ConverterProvider as an application-wide singleton (for your own code) but the library would not include the concept of an application-wide singleton. Correct?"

code reviews (73×) "How about a review when you get a chance?", "Can you rewrite the patch for the tip?"

Category #6: Status Enquiry

This category contains all questions that relate to the status (including resolution and priority) of a bug or its fix. Most of the questions are about the progress in general or whether a fix will make a certain version or build of the program. We also included questions about possible workarounds in the category because bugs with workarounds often have a lower priority.

→ *in total 80 questions, or 8.4% of all questions.*

questions about progress (28×) "Any progress on this one?"

questions about target version (19×) "Will this make M13? M14?"

questions about workaround (7×) "Can we see about recommending people upgrade viewpoint if they continue to crash with this?"

questions about related bugs (9×) "Now that bug 128586 is fixed, is this bug fixed, too?"

questions about resolution and priority (11×) "IF (sic) this is fixed on trunk, how come it's not marked RESOLVED-FIXED?"

reminders (6×) "Care to actually answer the question I asked?"

Category #7: Resolution

This category contains questions that ask whether a bug is resolved or whether it is still a problem. Typically, these questions are asked *after workarounds* have been mentioned, *after new builds*, or *after* longer periods of *inactivity* in the bug report. The latter kind of questions ensures that developers only spend their time on bugs that still matter to users. If users do not respond to such questions, the bugs get closed. The MOZILLA project even automated this process and posts *automated messages* to bug reports. However, closing bugs automatically after inactivity is not very popular among users [15].

→ *in total 68 questions, or 7.2% of all questions.*

after workaround (7×) "Does downgrading to 0.9.8 solve the problem?"

after new build (35×) "Do you still see the problem using FF2 or trunk?"

after inactivity (17×) "Has this been solved for any of you?"

automated message after inactivity (9×) "This is an automated message, with ID 'auto-resolve01'…"

Category #8: Process
This category contains questions about administrative tasks, best practices, and procedures.

→ in total 25 questions, or 2.6% of all questions.

administration (11×) "Do I have the authority to review code?", "Can you give me build engineer status/full shell access to download.eclipse.org?"

procedures (14×) "How should we track these kinds of issues?", "What should I do to get this through the process?"

5. STATISTICAL ANALYSIS

For each question and response (if available), we extracted time information from the bug database using the recorded bug and comment numbers (see Section 3). This allowed us to investigate the following aspects (correspond to dependent variables).[5]

- ***Question time.*** *When are questions asked in the life cycle of a bug report?*

  Out of the 940 questions, 11.3% were asked within one hour and 35.9% within a day after submission of the bug report (see Figure 1).

  Considering only the 773 questions in resolved bug reports, the majority of questions is asked in the first half of the bugs' lifetimes (65.8%).

- ***Response rate.*** *How many questions get answered?*

  Out of the 940 questions, 636 were responded to (67.66%). Note that this number considers the presence of a response only, it does not assess the quality of responses.

- ***Response time.*** *How long does it take to get a response?*

  Of the 636 questions with responses, 9.8% received responses within 10 minutes, 34.2% within the hour, and 79.4% within the day (see Figure 1).

We analysed the influence of the following factors (correspond to independent variables):

- *Category* — the card sort category to which the question belongs (for the eight categories, see Section 4).
- *Addressee* — question addressed at *developer* or *user*; for the analysis, we ignored questions which were addressed to both.
- *Topic* — question related to the *bug* or *fix*; for the analysis, we ignored questions that were related to both.
- *Project* — question was asked in ECLIPSE or MOZILLA.

Experimental Set-Up

We quantitatively investigated the effect of the above four factors on question time, response rate, and response time by adopting the following experimental set-up throughout our analysis:

[5]For the statistical analysis, we used the R statistical software [21] with the *car* package for the Anova tests.

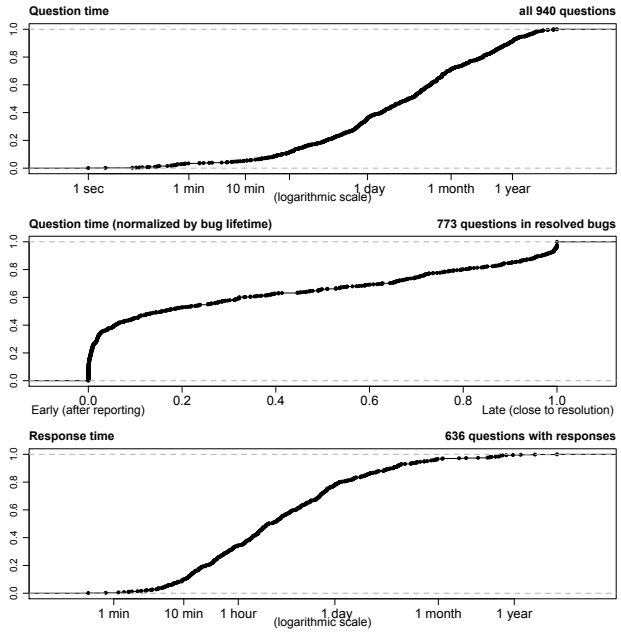

Figure 1. Distribution of question and response time.

Step 1. Find out which factors exert influence
We used a statistical modelling technique called Analysis of Variance (ANOVA) to measure which factors and interactions of factors influence the dependent variable. For example, in a three-way ANOVA with factors A, B, and C, the model tests for the effects of A, B, and C on the dependent variable, and also for the effects of the interaction terms AB, AC, BC, and ABC. We preferred Type II over Type III sum of squares for ANOVA because Type-II sums obey the principle of marginality and are better suited for unbalanced data [20].

For our analysis, we built two ANOVA models for each dependent variable. The first model included only *category* as the independent variable, while the second model included *addressee*, *topic*, and *project*. This separation was made to reduce chances of false negative errors (i.e., accepting the null hypothesis when it is false), which is a plausible effect of the high number of levels (discrete values taken by the factor) in the *category* factor. Factors (and interaction terms) found to have significant effects on the dependent variable are determined by observing the corresponding p-value which is a measure of arriving at the result purely by chance. Typically, a threshold value of $p \leq .05$ is considered to treat the factor significant and worthy of further analysis. Unless noted otherwise all results are significant at this level.

Step 2. Confirm and assess the effect of factors
ANOVA is followed by *post-hoc analyses* on statistically significant factors and interaction terms. In contrast to ANOVA, which only checks whether a factor significantly influences the dependent variable, post-hoc analyses investigate which specific levels of the factor are different from the others. For example, if an ANOVA yields that category influences response time, the post-hoc analysis investigates which of the eight levels of category actually have influence on response

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

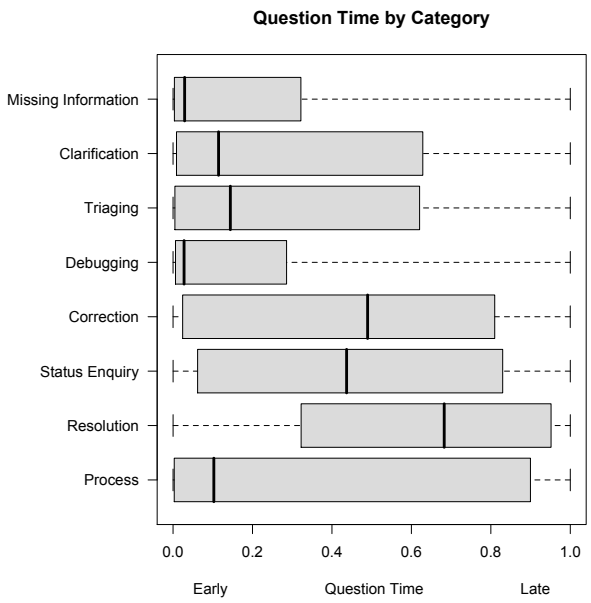

Figure 2. Boxplot of question time by category.

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).

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

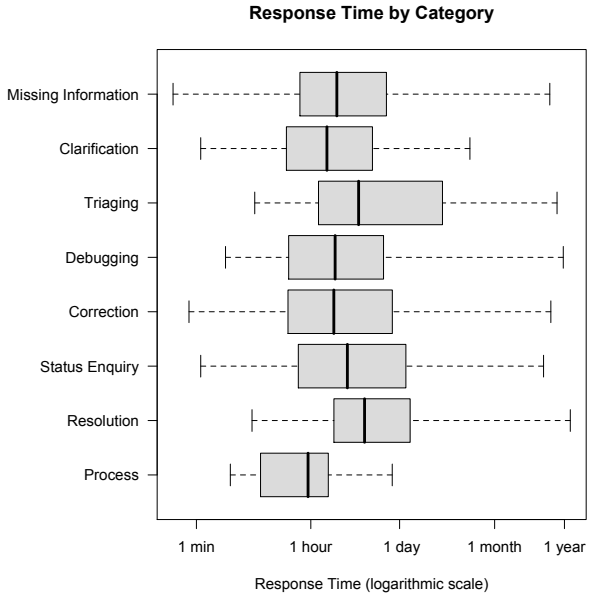

Figure 3. Boxplot of response time by category.

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 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.

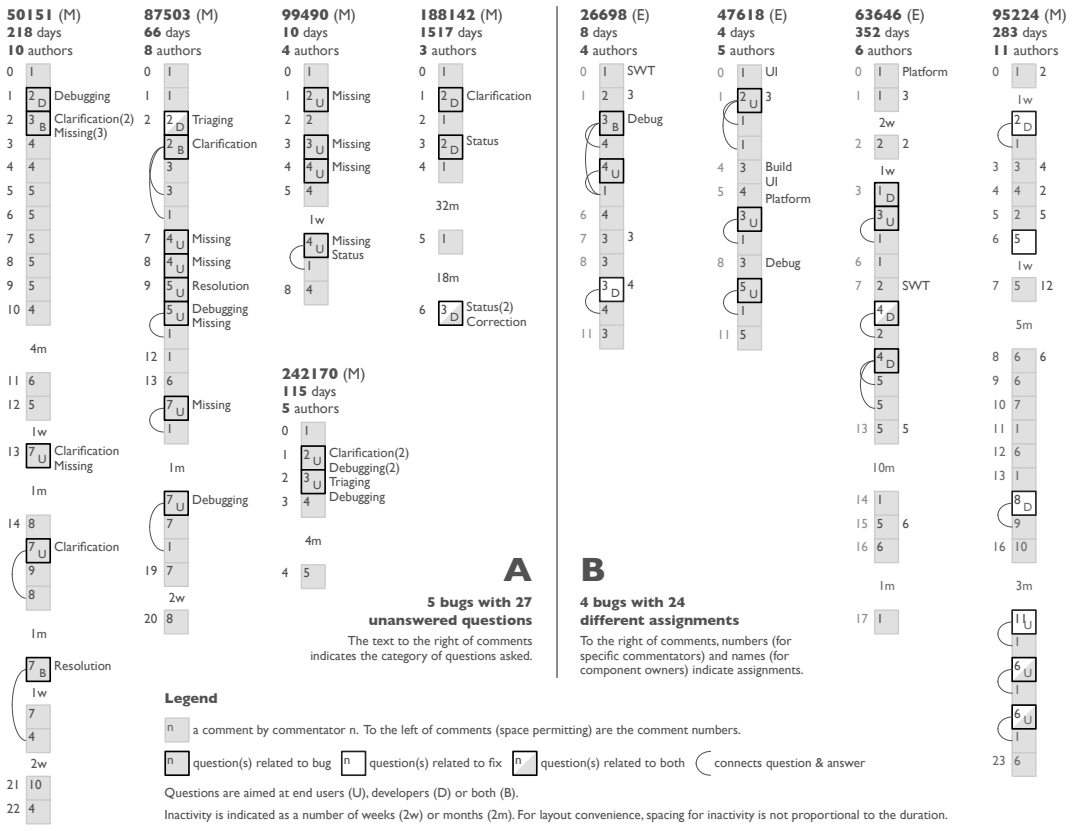

Figure 4. Illustration of discussion in nine bug reports.

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.