Quality of Bug Reports in Eclipse

Nicolas Bettenburg*
nicbet@st.cs.uni-sb.de

Sascha Just*
just@st.cs.uni-sb.de

Adrian Schröter*
adrian@st.cs.uni-sb.de

Cathrin Weiß*
weiss@st.cs.uni-sb.de

Rahul Premraj*§
premraj@cs.uni-sb.de

Thomas Zimmermann+§
tz@acm.org

* Department of Computer Science
Saarland University, Saarbrücken, Germany

+ Department of Computer Science
University of Calgary, Calgary, Alberta, Canada

ABSTRACT

The information in bug reports influences the speed at which bugs are fixed. However, bug reports differ in their quality of information. We conducted a survey among ECLIPSE developers to determine the information in reports that they widely used and the problems frequently encountered. Our results show that steps to reproduce and stack traces are most sought after by developers, while inaccurate steps to reproduce and incomplete information pose the largest hurdles. Surprisingly, developers are indifferent to bug duplicates. Such insight is useful to design new bug tracking tools that guide reporters at providing more helpful information. We also present a prototype of a quality-meter tool that measures the quality of bug reports by scanning its content.

1. INTRODUCTION

Bug reports are vital for any software development. They allow users to inform developers of problems that they encountered while using a software. A bug report typically contains a detailed description of a failure and occasionally pointers to the location that contains the fault (in form of patches or stack traces). However, bug reports often provide inadequate or incorrect information and developers have to face bugs with descriptions such as "wqqwqw" (ECLIPSE bug #145133), just "layout" (#52050) or "Logfin" (#178041). It is no surprise that developers are slowed down by poorly written bug reports because identifying the problem takes more time.

In this paper, we extract the notion of *quality of bug reports* from the perspective of developers. There are several factors that impact the quality of bug reports such as length of description, formatting, and presence of stack traces and attachments. In order to find out which ones matter the most, we asked 336 ECLIPSE developers to

1. *complete a survey* on important information in bug reports and problems faced with bug reports. We received a total of 48 responses to our survey (Section 2).
2. *rate bug reports* on a five-point Likert scale [10] from very poor to very good. We received a total of 397 votes on 100 randomly selected bug reports (Section 3).

By knowing what developers desire in bug reports, it is possible to provide tool support for reporters to furnish such information. For instance, one utility would be to provide immediate feedback on the quality of a bug report to users (like strength meters for passwords). As a first step in this direction, we developed a prototype tool called quZILLA that gauges the quality of bug reports (Section 4). We conclude this paper with a discussion on related work (Section 5) as well as possible improvements and future work (Section 6).

2. SURVEY ON BUG QUALITY

In order to collect facts on how developers use the information in bug reports and what problems they face, we decided to conduct an online survey among the developers of the ECLIPSE project.

2.1 Survey Design

For any survey, the response rate is crucial to draw generalizations from a population. Keeping a questionnaire short is one key to a high response rate. In our case, we aimed for a total time of five minutes, which we also advertised in the invitation email ("we would much appreciate five minutes of your time").

Selection of Participants.

The ECLIPSE bug database contains over 1,100 developers that are assigned to bug reports. We were interested in *experienced* developers because their experience is crucial to our research. We defined experienced as being assigned to at least fifty bug reports in the Eclipse project (as of June 13, 2007). We contacted 365 developers who met this criterion via email, 29 emails bounced back, leaving us with a population size of 336 developers. The results in this paper take into account the 48 responses that we received until July 22, 2007. The response rate of 14% is comparable to other Internet surveys, which typically range from 2% to 30%.

The Questionnaire.

Keeping the five minute rule in mind, we chose the following questions that we grouped into three parts (see also Figure 1):

Contents of bug reports. *Which items have developers previously used when fixing bugs? Which three items helped the most?*

Such insight greatly aids in guiding reporters to provide or even focus on information in bug reports that is most important to developers. We provided sixteen items selected on the basis of Eli Goldberg's bug writing guidelines [7]; or being standard fields in the BUGZILLA database.

§Contact authors are Rahul Premraj and Thomas Zimmermann.

Contents of bug reports.	Q1: Which of the following items have you previously used when fixing bugs? Q2: Which three items helped you the most?			
	❑ product	❑ hardware	❑ observed behavior	❑ screenshots
	❑ component	❑ operating system	❑ expected behavior	❑ code examples
	❑ version	❑ summary	❑ steps to reproduce	❑ error reports
	❑ severity	❑ build information	❑ stack traces	❑ test cases
Problems with bug reports.	Q3: Which of the following problems have you encountered when fixing bugs? Q4: Which three problems caused you most delay in fixing bugs?			
	You were given wrong:	There were errors in:	The reporter used:	Others:
	❑ product name	❑ code examples	❑ bad grammar	❑ duplicates
	❑ component name	❑ steps to reproduce	❑ unstructured text	❑ spam
	❑ version number	❑ test cases	❑ prose text	❑ incomplete information
	❑ hardware	❑ stack traces	❑ too long text	❑ viruses/worms
	❑ operating system		❑ non-technical language	
	❑ observed behavior		❑ no spellcheck	
	❑ expected behavior			
Comments.	Q5: Please feel free to share any interesting thoughts or experiences.			

Figure 1: The questionnaire presented to Eclipse developers.

Responders were free to check as many items for the first question (Q1), but at most three for the second question (Q2), thus indicating the importance of items.

Problems with bug reports. *Which problems have developers encountered when fixing bugs? Which three problems caused most delay in fixing bugs?*

Our motivation for this question was to find prominent obstacles that can be tackled in the future by a more cautious, and perhaps even automated, reporting of bugs.

Typical problems are reporters who accidentally provide incorrect information, for example an incorrect operating system.[1] Other problems in bug reports include poor use of language (ambiguity), bug duplicates, and incomplete information. Spam recently has become a problem, especially for the TRAC issue tracking system. We decided not to include the problem "incorrectly assigned to me" because bug reporters have little influence on the triaging of bugs.

In total, we provided twenty-one problems that developers could select. Again, responders were free to check as many items for the first question (Q3), but at most three for the second question (Q4).

Comments. *What are the thoughts and experiences of developers with the quality of bug reports?*

Parallelism between Questions.

In the first two parts of the survey, questions share the same items but have different limitations (select as many as you wish vs. the three most important). We will briefly explain the advantages of this parallelism on the example of Q1 and Q2.

1. *Consistency check.* When fixing bugs, all items that helped a developer the most (selected in Q2) *must* have been used previously (selected in Q1). If this is not the case and an item is selected in Q2 but not in Q1, the response is inconsistent and ignored.

[1]Did you know? In ECLIPSE, 205 bug reports were submitted for "Windows" but later re-assigned to "Linux".

2. *Importance of items.* We can additionally infer the importance of individual items. Let $Q1(i)$ be the number of times that item *i* was selected in question Q1, and $Q2(i)$ the same for question Q2. Then the importance of item *i* corresponds to the likelihood that item *i* is selected in Q2 when it is selected in Q1.

$$\mathit{Importance}(i) = \frac{Q2(i)}{Q1(i)}$$

2.2 Survey Results

In this section, we discuss our findings from the survey responses. To recall, we received a total of 48 responses until July 22, 2007. The results are summarized in Table 1. Responses for each item are annotated as bars ( ), which can be broken down into their constituents and interpreted as below:

	Response Category
	All 48 responses
	Number of responses for Q1
	Number of responses for Q1 and Q2
	Number of responses for Q1 and not Q2

The width of each bar accounts for the 48 responses received. The coloured part denotes the count of responses for the relevant item in question Q1; and the black part of the bar denotes the count of responses for the relevant item in question Q2 (and $Q1 \cap Q2$). Larger proportions of the black bar in comparison to the grey bar indicate higher importance of the corresponding item. The importance as a percentage is also listed in parentheses after every item in Table 1.

Used Contents of Bug Reports.

Most important were *steps to reproduce*; they were selected as useful by 47 responders (all but one) and 42 selected them to be among the three most useful elements in a bug report (importance 89%). Second and third in importance are *stack traces* and *screenshots*, again selected as important by a substantial majority of responders. Interesting surprises in these results are the relative unimportance of items such as *expected behaviour*, *summary* and mandatory fields such as *version*, *operating system*, and *product*.

We advise caution when interpreting these results: items with a low importance in our survey are not totally irrelevant because they still might be needed to understand, reproduce, or triage a bug.

Problems with Bug Reports.

The most severe problems for ECLIPSE developers are *errors in steps to reproduce* (importance 82%), *incomplete information* (importance 77%), and *wrong observed behaviour* (importance 66%).

A very interesting observation is for *duplicates*: while 31 responders encountered duplicates, only two ranked them as most problematic. Possibly, developers can easily recognize duplicated bug reports or sometimes even benefit by a different bug description.

The low occurrence of *spam* is not surprising: in BUGZILLA reporters have to register before they can submit bug reports, which successfully prevents spam. Lastly, *errors in stack traces* are infrequent, likely because they are copy-pasted into bug reports.

Developer Comments.

We received a total of fourteen comments by the responding developers. Most comments stressed the importance of a complete, correct, and clearly written bug description (see Table 1 for a selection). However, some comments revealed additional problems:

Different knowledge levels. *"In OSS, there is a big gap with the knowledge level of bug reporters. Some will include exact locations in the code to fix, while others just report a weird behavior that is difficult to reproduce."*

Netiquette. *"Another aspect is politeness and respect. If people open rude or sarcastic bugs, it doesn't help their chances of getting their issues addressed."*

Complicated steps to reproduce. This problem was pointed out by several developers: *"If the repro steps are so complex that they'll require more than an hour or so (max) just to set up would have to be quite serious before they'll get attention."* Another one: *"This is one of the greatest reasons that I postpone investigating a bug. . . if I have to install software that I don't normally run in order to see the bug."*

Also the developers pointed out situations in which bug reports get preferred treatment.

Human component. *"Another import thing is that devs know you (because you have filed bug reports before, you discussed with them on IRC, conferences, . . .)"*

Keen bug reporters. A developer wrote about reporters who identify offending code: *"I feel that I should at least put in the amount of effort that they did; it encourages this behavior."*

Serious bugs. *"For me it amounts to a consideration of 'how serious is this?' vs 'how long will it take me to find/fix it?'. Serious defects get prompt attention but less important or more obscure defects get attention based on the defect clarity."*

3. RATING BUG REPORTS

Once developers completed the questionnaire, we asked them to rate the quality of randomly selected bug reports. This part was voluntary and we therefore did not mention it in the invitation email.

Rating Infrastructure.

The rating system was inspired by rating sites on the Internet such as RateMyFace and HotOrNot. We drew a random sample of 100 bugs from the ECLIPSE bug database, which we presented one-by-one to the participants in a random order. They were required to read through the bug report and rate it on a five-point Likert scale ranging from very poor (1) to very good (5) (see Figure 2 for a screenshot). Once they rated a bug report, the screen showed the next bug report and a summary of the previously rated bug (on the left side). Developers could stop after any bug report. Having bug reports with a rating of quality, helps us with the following:

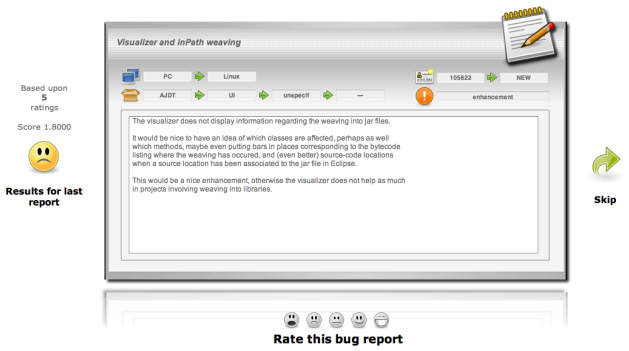

Figure 2: Screenshot of Rating Bugs' Quality Interface

1. They allow us to verify the results of the questionnaire on concrete examples, i.e., whether reports with highly desired elements are rated higher for their quality and vice versa.
2. We can use the rated bug reports to build and evaluate tools that predict the quality of bug reports. In Section 4, we present quZILLA, a prototype for such a tool.

Rating Results.

We received 397 votes for the sample of 100 bugs; 96 bug reports were rated by at least two unique responders. Table 2 lists the bug reports that had the highest and lowest average ratings. Bugs reports can be of exceptional quality, such as bug report #31021 for which both responders awarded a score of *very good*.

> *I20030205*
>
> *Run the following example. Double click on a tree item and notice that it does not expand.*
>
> *Comment out the Selection listener and now double click on any tree item and notice that it expands.*
>
> *public static void main(String[] args) {*
> *Display display = new Display();*
> *Shell shell = new Shell(display);*
> [. . .] (21 lines of code removed)
> *display.dispose();*
> *}*
>
> (ECLIPSE bug report #31021)

On the other hand, bug report #175222 with an average score of 1.5 is of fairly poor quality.

> *I wand to create a new plugin in Eclipse using CDT. Shall it possible. I had made a R&D in eclipse documentation. I had get an idea about create a plugin using Java. But i wand to create a new plugin (user defined plugin) using CDT. After that I wand to impliment it in my programe. If it possible?. Any one can help me please...*
>
> (ECLIPSE bug report #175722)

Contents of bug reports.

product (0%)	hardware (0%)	observed behavior (27%)	screenshots (40%)
component (0%)	operating system (0%)	expected behavior (7%)	code examples (23%)
version (2%)	summary (8%)	steps to reproduce (89%)	error reports (9%)
severity (0%)	build information (8%)	stack traces (77%)	test cases (28%)

Problems with bug reports.

You were given wrong:	There were errors in:	The reporter used:	Others:
product name (0%)	code examples (18%)	bad grammar (17%)	duplicates (6%)
component name (9%)	steps to reproduce (82%)	unstructured text (26%)	spam (0%)
version number (28%)	test cases (28%)	prose text (8%)	incomplete information (77%)
hardware (0%)	stack traces (0%)	too long text (18%)	viruses/worms
operating system (25%)		non-technical language (21%)	
observed behavior (66%)		no spellcheck (0%)	
expected behavior (22%)			

Selected comments.

Incomplete information is the biggest problem. Usually if someone provides steps to reproduce, stack traces, observed behavior, etc., it is helpful even if the grammar or language is not so great.

—

The most annoying problem is too brief bug description.

—

Using Bug report template for bug Description would be helpful, I think.

The most important info that a reporter can provide is a way to reliably reproduce the problem.

—

Really good bug reports will:
1) describe the difference between the observed and expected behaviors
2) have -clear-, fairly simple, repro steps
3) Provide stack traces and/or screen shots supporting the above info.

Table 1: The results of the questionnaire. Overall 48 out of 336 contacted ECLIPSE developers responded.

Bug Report	Votes	Rating
Tree - Selection listener stops default expansion (#31021)	2	5.00
JControlModel "eats up" exceptions (#38087)	4	4.75
Search - Type names are lost [search] (#42481)	4	4.50
150M1 withincode type pattern exception (#83875)	4	4.50
ToolItem leaks Images (#28361)	5	4.40
...	...	...
Outline view should [...] show all project symbols (#108759)	2	2.00
Selection count not updated (#95279)	3	2.00
Pref Page [...] Restore Defaults button does nothing (#51558)	5	1.80
[...]<Incorrect /missing screen capture> (#99885)	4	1.75
Create a new plugin using CDT. (#175222)	6	1.50

Table 2: Developers rated the quality of ECLIPSE bug reports.

Overall, the agreement among developers on the quality of individual bug reports was rather strong. For only 22 bugs, the ratings awarded by developers had a standard deviation greater than one (max. being 1.5).

4. THE QUZILLA TOOL

Besides asking developers about bug quality issues, we developed a prototype tool — 'quZILLA', to automatically measure the description quality of a bug report. quZILLA is based on the ECLIPSE guidelines on how to write good bug reports [7]. The tool is implemented in PYTHON and uses the NLTK toolkit for Natural Language Processing (NLP) [11]. Bug descriptions are given as input, which are first preprocessed by tokenization (splitting the text into single words/tokens) and stemming (reducing words to their stem form). Then, quZILLA determines a quality score using the following criteria:

- *How readable is the bug description?* We regard the readibility of a bug report high if it contains a blank line every three to six lines. If so, it achieves 20 points. We considered these numbers useful while reading technical short instructions, such that texts do not appear as one huge block. Additionally, we considered enumerations to enhance readibility. We determined enumerations in the reports by checking whether several subsequent lines contain a character like "`-`", "`*`" or enumerations such as "`1), 2)`" in the beginning. If present, the bug report was awarded another 30 points.

- *Are specific keywords present?* Some keywords hint at important information, like *reproduce* may hint at the presence of *steps to reproduce*. Other examples of keywords include *stack trace*, *screenshot*, *exception*, *behaviour*, *runtime*, and *violation*. Depending upon the number of keywords present, up to 60 points are awarded to the report.

- *Does the bug description contain code examples?* This is measured by checking for keywords that may hint at program code. We award 15 points if any code keyword is found.

The maximum score a bug report can achieve is 125. For evaluation, we compared the quality score delivered by quZILLA for the bugs presented to developers in the survey to the scores awarded by developers themselves.

Our results are presented in Table 3. The rows represent scores determined by quZILLA (predicted) and the columns represent average developer ratings (observed). We mapped the scores by quZILLA to the Likert scale keeping in mind that the prototype currently computes the final score additively for information items individually instead of looking for classes of information. The results across the diagonal (dark gray background) currently show a fair degree of agreement between quZILLA and developers accounting for 42% of the bugs. Additionally, for over 90% bugs, the tool and developers agreed within one interval from each other (light gray background). The results warrant further improvements to quZILLA which can be achieved by improving the detection of information in the reports and reviewing the scoring mechanism.

5. RELATED WORK

To our knowledge, no other work has specifically studied the quality of bug reports or suggested a quality-meter tool for bug reports.

Several studies used bug reports to automatically assign developers to bug reports [2, 4], assign locations to bug reports [3], track features over time [6], recognize bug duplicates [5, 12], and predict effort for bug reports [13]. All these approaches should benefit by

Predicted		Observed very poor	poor	medium	good	very good
very poor	$[0, 4]$	0	0	0	0	0
poor	$[5, 12]$	0	1	2	1	0
medium	$[13, 33]$	0	5	21	12	0
good	$[34, 90]$	1	5	20	18	3
very good	$[91, 125]$	0	0	3	6	2

Table 3: The results of the classification by quZILLA compared to the developer rating.

our measure for the quality of bug reports since training only with high-quality bug reports will likely improve their predictions.

In order to inform the design of new bug reporting tools, Ko et al. [9] conducted a linguistic analysis of the titles of bug reports. They observed a large degree of regularity and a substantial number of references to visible software entities, physical devices, or user actions. Their results suggest that future bug tracking systems should collect data in a more structured way.

In 2004, Antoniol et al. [1] pointed out the lack of integration between version archives and bug databases. Providing such an integration allows queries to locate the most faulty methods in a system. While the lack of integration was problematic a few years ago, things have changed in the meantime: the Mylar tool by Kersten and Murphy [8] allows to attach a task context to bug reports so that changes can be tracked on a very fine-grained level.

6. CONCLUSION AND CONSEQUENCES

Well written bug reports are likely to get more attention among developers than poorly written ones. In order to get a notion of *bug report quality* from the point of view of developers, we conducted a survey among ECLIPSE developers. The results suggest that steps to reproduce and stack traces are most useful in bug reports. The most harmful problems encountered by developers are errors in steps to reproduce, incomplete information, and wrong observed behavior. Surprisingly, bug duplicates are encountered often but not considered as harmful by developers.

Additionally, the ECLIPSE developers classified 100 bug reports on a scale from one (poor quality) to five (excellent quality). Based on the results from the survey, we constructed a measure of bug report quality that was able to classify 42% of the bugs correctly. In the long term, having an automatic measure of bug report quality can ensure that new bug reports meet a certain quality level and serve as a data cleaning technique for research on bug reports. Our future work is thus as follows:

Collect more data. We are currently posting our survey to the developers of the APACHE, JBOSS, and MOZILLA projects. Additionally, we will target bug reporters in order to find out which information is difficult to provide. By combining the results from both surveys, we might be able to propose new tools for bug reporting.

Immediate feedback to reporter. When a user is reporting a new bug, we can give her immediate feedback on the quality of her report in form of a *quality meter*: while the user is typing a reports, the color of the meter indicates its quality. By taking historic data into account, we can encourage the reporter further: "Bug reports with stack traces have been fixed twice as fast," or "Add a screenshot to increase your quality by 20."

To learn more about our work in mining software archives, visit

`http://www.softevo.org/`

Acknowledgments. Many thanks to Harald Gall, Christian Lindig, Stephan Neuhaus, Andreas Zeller, and the anonymous ETX reviewers for their valuable discussions and helpful suggestions on earlier revisions of this paper. A special thanks to all ECLIPSE developers who responded to our survey. When this research was carried out, Thomas Zimmermann was with Saarland University and funded by the DFG Research Training Group "Performance Guarantees for Computer Systems".

7. REFERENCES

[1] G. Antoniol, H. Gall, M. D. Penta, and M. Pinzger. Mozilla: Closing the circle. Technical Report TUV-1841-2004-05, Technical University of Vienna, 2004.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proceeding of the 28th International Conference on Software Engineering*, pages 361–370, 2006.

[3] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. In *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 105–111, 2006.

[4] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1767–1772, 2006.

[5] D. Cubranic and G. C. Murphy. Automatic bug triage using text categorization. In *SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, pages 92–97, 2004.

[6] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003), 13-16 November 2003, Victoria, Canada*, pages 90–101, 2003.

[7] E. Goldberg. Bug writing guidelines. https://bugs.eclipse.org/bugs/bugwritinghelp.html. Last accessed 2007-08-04.

[8] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2006)*, pages 1–11, 2006.

[9] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006)*, pages 127–134, 2006.

[10] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.

[11] NLTK toolkit for natural language processing. http://nltk.sourceforge.net. Last accessed 2007-08-04.

[12] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 499–510, 2007.

[13] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007.