Understanding Expressions of Unwanted Behaviors in Open Bug Reporting

Parmit K. Chilana, Andrew J. Ko and Jacob O. Wobbrock
The Information School, DUB Group, University of Washington
{pchilana, ajko, wobbrock}@uw.edu

Abstract

Open bug reporting allows end-users to express a vast array of unwanted software behaviors. However, users' expectations often clash with developers' implementation intents. We created a classification of seven common expectation violations cited by end-users in bug report descriptions and applied it to 1,000 bug reports from the Mozilla project. Our results show that users largely described bugs as violations of their own personal expectations, of specifications, or of the user community's expectations. We found a correlation between a reporter's expression of which expectation was being violated and whether or not the bug would eventually be fixed. Specifically, when bugs were expressed as violations of community expectations rather than personal expectations, they had a better chance of being fixed.

1. Introduction

Popular Open Source Software (OSS) projects such as Mozilla are inundated with hundreds of bug reports every day from end-users around the world. One way that the OSS community copes with this daily wave of issues is to separate them roughly into two categories: (1) problems that violate developers' intents, and (2) everything else, including feature requests, help requests, issues out of a team's control, among others [1,12].

Of course, most end-users know little about developers' intents: they simply know that an application did something unwanted and often users report that unwanted behavior as a bug. But what is an "unwanted" behavior from the user's perspective? And how are users notions of "unwanted" software behaviors related to developers' intents? And how does this clash between users' expectations and developers' intents affect which reported bugs are addressed? This paper investigates these questions, complementing recent studies of bug reporting from developers' perspectives [2,3,7]. We analyzed user contributions to the bug reporting process in a prior study [9], but in this paper, we focus on bug report topics that emerge from user descriptions.

Our approach was to randomly sample bug reports from Mozilla's Bugzilla repository, analyzing unwanted behaviors described in the report titles and descriptions. We found that in describing unwanted behaviors, reporters implicitly referred to one or more common classes of expectations that had been violated, including the reporter's personal experiences or the practices of the user community at large. From this initial analysis, we extracted a classification scheme of seven common *expectation violations* and applied it to 1,000 Mozilla bug reports. Using these classifications, we analyzed the relationship between the expectations identified in each report and whether the report was eventually resolved as *fixed*. Our key findings reveal that, at least in the Mozilla project, whether a bug is fixed depends largely on whether the reporter explicitly refers to the developers' intents or to the expectations of the Mozilla user community. All other forms of expectations receive little attention.

This work contributes: (1) a conceptualization of unwanted behaviors described in bug reports as a violation of expectations, (2) a classification scheme for capturing the different types of expectation violations, (3) an analysis of expectation sources from a large sample of Mozilla reports, and (4) empirical findings that show the relationship between the expectation source identified in a bug report and the report's final resolution. We conclude by discussing the implications of our classification scheme on open bug reporting and bug reporting tools, highlighting some ways that OSS communities can better leverage contributions from end-users.

2. Method

To study and classify unwanted behaviors described in bug reports, we gathered data from the Bugzilla repository of the Mozilla project. We chose to study the Mozilla project because of its large user base and its reputation as a user-centered open source project. We downloaded all available Mozilla bug reports, 496,766 in total, on August 14, 2009 using standard HTTP queries. Since we were interested in whether or not a bug was eventually fixed, we did not include any bug reports that were still open. We focused on bugs that had been reproduced and decided upon by selecting bugs marked as CLOSED, RESOLVED, or VERIFIED in Bugzilla. This filtering criteria resulted in 420,005 reports. We wrote Perl scripts for our initial exploration of the bug report data and for computing some variables of interest.

We next describe the method that we used to classify unwanted behaviors in bug descriptions and

our application of the resulting classification scheme onto a sample of 1000 reports.

2.1 Classification of Unwanted Behaviors

We first selected a sample of 50 bug reports and analyzed unwanted behaviors described in the report's titles and descriptions. Through this analysis, we found that users implicitly referred to different expectations that had been violated as they described unwanted behaviors. We decided that the *source of expectation* a reporter believed was violated was a potentially interesting and important variable. To operationalize source of expectation, we employed an inductive analysis approach [6], classifying and reclassifying our descriptions of the different sources of expectations. The first author independently examined 3 sets of randomly selected bug reports (100 reports each), generating descriptions of what was being violated in the bug report titles and descriptions. These descriptions converged on a single coding scheme after numerous iterations and discussions with the other authors.

Our classification of the different sources of expectations consisted of seven codes. The first three codes represent more conventional notions of a bug as a violation of developer intent:

Runtime logic. Explicit violations of some runtime expectation, including errors, warnings, assertion violations, crashes, and hangs (e.g., *"…scary deadlock assertions exiting mozilla after referencing nsInstallTrigger…").*

Specification. An agreed upon functional requirement among the developers (e.g., "*There's an incorrectly placed PR_MAX in the code for pref width distribution of colspanning cells.").*

Standards. Specifications shared by the industry in which the application is deployed (e.g., *"'codebase' attribute of the HTML 4.0 OBJECT element is not supported…"*).

The remaining four categories refer to other sources of expectations, outside the scope of the implementation, developer community, or industry:

Reporter expectations. A reporter's personal perspective about what the system should do (e.g., *"Every time I Sort By Name by Bookmarks Firefox sorts and closes my Bookmark menu… Why does it do this??").*

Community expectations. A reporter's belief about a "typical" user's expectations, including specific references to *user*, *users*, *user interface*, or *usability*. (e.g., *"The preference to not show the tab bar when only one tab is open could be set to false by default. This would at least alert a new user to the possibility that tabs exist) The old tabbed browsing preferences could be returned.").*

Genre conventions. References to applications with similar functionality; allusions to how a specific feature behaves for the same action. (e.g., *"Firefox does not limit the slideshow horizontal size to the window width. The same source works correctly in IE.").*

Prior behavior: References to the prior desirable behavior of the system (e.g., *"The latest version of Firefox only imports one certificate from each file. I used to import all certificates previously.").*

While these categories may not be exhaustive, they did capture the full range of expectation violations found in our sample.

2.2 Sampling and Analysis

To test our classification, we next selected a uniform random sample of 1,000 bug reports from our data set, excluding those used to develop the classification. The first author applied the coding scheme described above to our sample. To assess the reliability of the coding scheme, the second author coded a subset of 100 reports. For this subset, there was a 78% agreement on issue types between the two coders (κ=0.62). Finally, note that a small portion of bugs in our sample (n=25) did not fit our coding scheme because they described meta-issues about the bug reporting process; these were excluded from our analyses.

Next, we explored the association between *source of expectation* and *bug resolution*. Upon our initial analysis, we observed that 30.07% were marked DUPLICATE. We then further analyzed the resolution of these DUPLICATE bugs to determine their final resolution flags. For simplicity, we marked the bugs that were fixed as DUPLICATE_FIXED and grouped all other resolution flags as DUPLICATE_NOTFIXED. Table 1 shows the distribution of bug resolution flags in our sample, and a brief description of each resolution status.[1]

FIXED fixed by a check-in	40.00%
DUPLICATE_FIXED duplicate of another bug and was fixed	16.40%
DUPLICATE_NOTFIXED duplicate of another bug and was not fixed	13.70%
WORKSFORME cannot be reproduced	13.40%
INVALID observed behavior is the intended behavior	9.80%
WONTFIX valid but cannot be fixed	3.50%
EXPIRED expired after a period of inactivity	1.80%
INCOMPLETE steps to reproduce are not complete	1.40%

Table 1: Distribution of Resolution Flags in Our Sample

[1] Source of Mozilla-specific bug resolution definitions: https://developer.mozilla.org/en/What_to_do_and_what_not_to_do_in_Bugzilla

3. Results

We now report our main findings: (1) the distribution resulting after applying our coding scheme to a sample of 1,000 bugs and (2) the correlation between the *source of expectation* and *bug resolution*.

Figure 1: Distribution of sources of expectations.

Figure 1 shows the distribution of sources of expectations violated in our sample of 1,000 bugs. Clearly, the largest portion of our bugs in our sample were **reporter expectations,** which referred to violations of reporter's personal expectations (*n*=337). Violations of **runtime logic** (*n*=195) and **specification** (*n*=177) were the next largest groups. The remaining groups each accounted for less than 10% of the sample and were distributed as follows: **community expectations** (*n*=85), **genre conventions** (*n*=71), **prior behavior** (*n*=69), and **standards** (*n*=41).

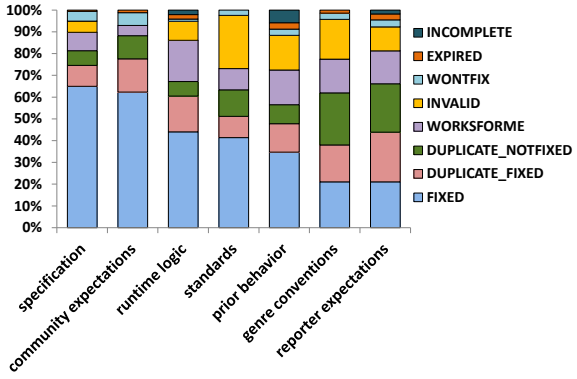

Figure 2: Distribution of resolution categories for sources of expectations.

To assess how the source of expectation affected bug resolution, we performed multinomial regression with *source of expectation* as a nominal predictor and bug resolution as a nominal outcome. We found that the *source of expectation* had a significant effect on bug resolution ($\chi^2(7, N=1000) = 35.8$, $p<.001$). Figure 2 shows the relationship between the *source of expectation* and *bug resolution* categories. As shown, over half of the bugs that were about violations of **specification** and **community expectations** were FIXED. Although **reporter expectations** occupied the largest proportion in our sample distribution (Figure 1), only about 20% of these reports were first resolved as FIXED—about half of these bugs were initially marked as DUPLICATE and only about 20% of the duplicate bugs were eventually FIXED. Bugs about **standards**, **genre conventions**, and **prior behavior** were more likely to get marked INVALID, meaning that the developers considered the actual behavior to be the intended behavior and not violations.

Furthermore, the distribution of FIXED bugs shows that when users identified unwanted behaviors that were violations of **specification**, **community expectations**, or **runtime logic**, they were more successful in getting their bugs resolved as FIXED. On the other hand, when users cited personal experiences only, or conventions in competing systems, they achieved little success.

4. Discussion

Our study contributes a detailed articulation of the unwanted behaviors that users describe in bug reports. Our analysis shows that there is a correlation between a user's expression of whose expectation is being violated and whether or not the bug will be fixed. Although, our results are limited to Mozilla, below we discuss implications of our classification scheme on understanding end-user bug reporting behavior and augmenting the design of bug reporting tools.

4.1 Implications for Expanding the Notion of a Bug

First, our findings reveal the limitations of simple divisions between unintended behavior and feature requests, expanding the notion of a bug to a wide variety of sources of user expectations. If we look at the bugs in our sample from the perspective of binary classifications (*i.e.*,[1]), real "bugs" (the **specification** and **runtime logic** violations in our scheme) only accounted for about 37.2% of our sample. By this measure, over 60% of the other bug reports were simply "non-bugs." But through our classification, we learned that these "non-bugs" encompassed a range of unwanted behaviors that violated the users' idiosyncratic personal expectations, exposure to previous versions of a system or use of other similar systems. The developers were in fact responsive to fixing many of these "non-bugs" provided that they were expressed as violations of the user community's expectations. (These findings also contrast the extant belief (*cf.* [5,10]) that OSS developers tend to focus only on issues relevant to errors in code or functionality problems.)

Our results open an intriguing question: can users "game" open bug reporting by articulating a problem in community terms? Or do developers eventually

uncover the reality of an issue? It appears that *what users write* and *what the real issue is* are two dimensions of a bug report. Future studies should investigate how an issue's phrasing really affects the outcome of a report. Our results suggest that the answer to this question will depend on the *source of expectation*. For example, it appeared that many users had a difficult time accurately interpreting the meaning of HTML specifications, which led to several invalid reports. However, in the case of **reporter expectations**, there may be many common, critical usability issues behind individual issue descriptions that are never discovered, simply because of how they are phrased.

4.2 Implications for Bug Reporting Tools

With the current design of open bug reporting tools, it is likely that end-users will continue to submit a large number of isolated idiosyncratic descriptions of unwanted behaviors, most of which will not get fixed. But if 10,000 such idiosyncratic descriptions were to point to the same issue, how could we redesign bug reporting tools so that the community impact of such an issue is more obvious and the chances of that issue being resolved are increased?

Current focus on enhancing bug tracking tools has been on improving the quality of the bug report [3,8], and the information exchange between end-users and developers [4]. But these improvements largely benefit developers. To better leverage user participation in open bug reporting, our results suggest that bug reporting tools should provide the user with: (1) more concrete ways to express a range of unwanted behaviors, and (2) some form of feedback about the extent to which the reported issue also affects the larger user community. For example, if tools could automatically identify violations of personal expectations in bug report descriptions, users could learn up front that their bugs are not likely to get fixed. This feedback would perhaps encourage users to refine their reports or investigate other ways of resolving their individual issue.

Also, if end-users have more concrete ways of expressing unwanted behaviors, and bug reporting tools can be designed to aggregate these in a meaningful way, OSS developers could have a rich view of community impact and be able to make more informed software evolution decisions.

4.3. Conclusion

Open bug reports serve as a forum for users to communicate with developers and express a range of unwanted behaviors, as seen by our classification scheme. Our results illustrate how articulation of community impact can allow users to have more success in getting problems resolved. Our current analysis did not take into account possible confounds that could affect bug resolution, which we plan to include in future work. It would be particularly interesting to examine other factors that influence reporters to describe bugs as violations of their personal expectations. For example, reporters who have not yet diagnosed their problems may just tend to report non-issues and tend to explain things in personal terms instead of community terms.

Acknowledgements

This material is based in part upon work supported by the National Science Foundation under Grant Number CCF-0952733 and a doctoral fellowship by the Social Sciences and Humanities Research Council of Canada.

References

[1] Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., and Guéhéneuc, Y. Is it a bug or an enhancement?: a text-based approach to classify change requests. *Proc Conf of the Center for Advanced Studies on Collaborative Research,* (2008).

[2] Anvik, J., Hiew, L., and Murphy, G. Who should fix this bug? *Proc ICSE,* (2006), 361-370.

[3] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. What makes a good bug report? *Proc FSE* (2008), 308-318.

[4] Breu, S., Premraj, R., Sillito, J., and Zimmermann, T. Information needs in bug reports: improving cooperation between developers and users. *Proc CSCW,* (2010).

[5] Dalle, J., den Besten, M., and Masmoudi, H. Channeling Firefox Developers: Mom and Dad Aren't Happy Yet. In *Open Source Development, Communities and Quality*. Springer, Boston, (2008), 265-271.

[6] Glaser, B.G. *Basics of grounded theory analysis: emergence vs forcing*. Sociology:Mill Valley, CA, (1992).

[7] Glerum, K., Kinshumann, K., Greenberg, S., et al. Debugging in the (Very) Large: Ten Years of Implementation and Experience. *Proc SOSP* (2009).

[8] Just, S., Premraj, R., and Zimmermann, T. Towards the next generation of bug tracking systems. *Proc VL/HCC*, (2008), 82–85.

[9] Ko, A.J. and Chilana, P.K. How power users help and hinder open bug reporting. *Proc CHI'10*, 1665-1674.

[10] Lakhani, K. and Wolf, R.G. Why hackers do what they do: Understanding motivation and effort in free/open source software projects. *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, (2005), 3–21.

[11] Nichols, D. and Twidale, M. The usability of open source software. *First Monday 8*, (2003), 1-6.

[12] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. How Long will it Take to Fix This Bug? *Proc ICSE Workshop on Mining Software Repositories*, (2007).