How Power Users Help and Hinder Open Bug Reporting

Amy J. Ko and Parmit K. Chilana
The Information School | DUB Group
University of Washington
{ajko, pchilana}@uw.edu

ABSTRACT
Many power users that contribute to open source projects have no intention of becoming regular contributors; they just want a bug fixed or a feature implemented. How often do these users participate in open source projects and what do they contribute? To investigate these questions, we analyzed the reports of Mozilla contributors who reported problems but were never assigned problems to fix. These analyses revealed that over 11 years and millions of reports, most of these 150,000 users reported non-issues that devolved into technical support, redundant reports with little new information, or narrow, expert feature requests. Reports that did lead to changes were reported by a comparably small group of experienced, frequent reporters, mostly before the release of Firefox 1. These results suggest that the primary value of open bug reporting is in recruiting talented reporters, and not in deriving value from the masses.

Categories and Subject Descriptors
H.5.3 [Group and Organization Interfaces], D.2.5 Testing and Debugging.

General Terms
Design, Human Factors.

Keywords
Open source software, Bugzilla, Mozilla, Firefox

INTRODUCTION
Of all aspects of open source software (OSS) development, one of the most user-centered is that *anyone* can report problems that they find [12]. While in all likelihood, it is mostly *power* users who report problems, this idea still has fascinating implications for HCI, user-centered design, and the dialog between developers and user communities. For example, not only can power users report bugs such as crashes and data loss, but they can also identify design issues and advocate for less technical users they represent in their jobs. Moreover, they can provide this feedback to developers directly, rather than through traditionally slow and opaque technical support channels.

With many OSS projects now more than a decade old, we can finally ask the question: how good are power users at actually reporting software problems? Do tech savvy early adopters identify issues that core OSS developers do not? When these users report problems? And do any "regular" users report problems? Prior work has repeatedly shown *that* users write reports [2,11], and that some of these users become active OSS developers [8,9], but no studies have considered the long-term value of these crowdsourcing quality assurance to the public. For any software team considering whether to maintain an open bug reporting community, answering these cost/benefit questions is of critical importance. Moreover, through such an analysis, we may identify better ways to *design* open bug reporting tools to maximize the values that users provide.

To this end, we contribute a comprehensive analysis of the bug report contributions to Mozilla project, which is well-known for its transparent process and public participation in the development of the Firefox web browser. From its bug report repository of nearly a half million bug reports, we first take a bird's eye, quantitative view of user contributions, and then a more detailed, qualitative look at user comments and developers' reactions to them. Through these analyses, we found that over the past 11 years, most of the 150,000 users in this group reported either non-issues that devolved into technical support, redundant reports with little new information, or narrow expert feature requests. Reports that *did* lead to changes were reported by a comparably small group of about 8,000 experienced, frequent reporters and largely before the release of Firefox 1.

These results suggest that the value to be obtained from open bug reporting repositories is primarily in recruiting and retaining talented developers and reporters, and not in deriving value from the masses. This is in contrast to other forms of crowdsourcing, where much of the value is in the long tail of contributions. We discuss the implications of these findings on software development and on the design of bug reporting tools, and suggest several ways that open bug reporting tools might be redesigned to incentivize more helpful user contributions.

RELATED WORK
While there has been a considerable study of open source software development, little has focused specifically on the interactions between users and developers in bug reporting. The study most closely related to ours surveyed and compared the opinions of open source developers and bug reporters in Mozilla and other communities, finding a mismatch in what content each group viewed as important in a bug report [2]. In free response questions, developers had several insights: "there is a big gap with the knowledge level of bug reporters," "If people open rude or sarcastic bugs, it doesn't help their chances of getting their issues addressed", "Bugs are often used to debate the relative importance of various issues. This debate tends to spam the bugs with

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

various use cases and discussions [...] making it harder to locate the technical arguments often necessary for fixing the bugs," and, "Well known reporters usually get more consideration than unknown reporters, assuming the reporter has a pretty good history in bug reporting." Our study is an opportunity to evaluate these claims quantitatively and to discover new trends unknown to OSS developers.

Other work has considered interactions between users and developers in other contexts. For example, Hendry's analysis of the role of users in development of *del.icio.us* showed that many users offered creative input, offering appeals to personal experience, scenario, and observed use [6]. Another study considered help seeking in open source forums, finding a variety of "me too" messages that provided emotional support and rapid problem diagnosis [14]. These studies demonstrate that in some contexts, interactions between developers and users can be mutually beneficial.

The majority of other research on OSS communities focuses on the coordination necessary to develop software. For example, one of the most commonly cited findings is that open source communities are like "onions," with increasingly large groups of less technically savvy contributors [4]. At the core are a small team of (often collocated) developers; around them are volunteer developers who contribute regularly; beyond this group are bug reporters, source code readers, and finally passive users of the project's software. Theoretically, each of these successive groups is an order an of magnitude larger than the last.

As part of this model, several have studied the transitions that users make to become active developers. For example, van Krogh et al. studied the *Freenet* community with interviews, finding that the core community of developers made explicit rules that must be followed to join the community, based on merit [9] Similarly, Herraiz et al. studied the *GNOME* version control system, finding that hired developers follow similar patterns as volunteers to the project, but acquire status and reputation faster than volunteers [7]. Other studies demonstrate most core developers in the *Mozilla*, *Apache*, and *NetBeans* projects are full-time employees of corporate or non-profit organizations, not volunteers [8], and that volunteers are primarily distinguished by their success at several merit-based rites of passage [5]. Our study differs from these in that we consider bug reporters who contribute infrequently or only once.

Other research focuses more on coordination aspects of open source communities. For example, Mockus et al. contrasted the early history of the Apache and Mozilla projects, showing that *Apache's* coordination concerns were reduced by virtue of its architecture, allowing developers to contribute quickly and independently [11]. In contrast, Mozilla had significant interdependence between its modules, leading to the notion of *module owners*. Mockus et al. argue that this more formal means of coordination led Mozilla to delegate much of the bug fixing and finding to the user community. Several researchers have studied the manifestation of these coordination challenges in bug reports [13], mailing lists [1], email and CVS [15], and forums [10]. These studies show that OSS cultures are often biased towards *action* first and coordination later. We know of no studies that consider how users may help or hinder these coordination efforts.

In summary, prior work shows that open source communities use lightweight communication tools to coordinate and rites and reputation to foster a community of trusted developers. The primary question posed in our study is to what extent open source communities benefit from the *broader* user population through bug reporting and how bug reporting tools might be improved to encourage helpful contributions.

METHOD

We divide our assessment of user contributions to the Mozilla bug repository into three major sections:

1) Separating contributors into four categories of *core* and *active* developers, *reporters*, and *users*.
2) Analyzing the *outcomes* of reports written by different contributor groups.
3) Analyzing user and reporter *comments* in both routine and contentious reports, and developers responses.

Our choice of Mozilla was based on several factors. It is one of the most successful *user-facing* open source communities, and was likely to exhibit less of the hacker culture attributed to some OSS projects. Mozilla products are also respected as *usable* software, partly due to the fact that the Mozilla corporation employs user experience designers. Furthermore, at over a decade old, we would be able to analyze user contributions to Mozilla products over time and multiple substantial releases, revealing how participation changes over the course of years. Finally, while Mozilla involves several different projects, we chose to analyze them together because of their shared source code foundation and contributor communities.

Our data set was the Mozilla Bugzilla bug report repository (bugzilla.mozilla.org). We downloaded all bug reports as XML using standard HTTP queries on August 14th and 15th, 2009. This data set included 496,766 reports, with creation dates as early as September of 1994. Not all reports were publicly accessible; 14,049 were only available to contributors with special permissions. Much of the data we report on in this paper was extracted, aggregated, and joined with Perl scripts and special care was taken to test their correctness through extensive error handling and assertions (for example, tests for missing values, invalid nominal data, and incidental case mismatch).

In terms of the data reported in this paper, all distributions in followed power law distributions (as with most social systems), unless otherwise noted. Consequently, we primarily report medians instead of means. All statistical inferences were non-parametric, unless otherwise noted. We primarily used chi-squared tests, Wilcoxon rank sums tests with chi-squared approximations (abbreviated RS), and multinominal logistic regressions with chi-squared approximations (abbreviated MLR), all performed in JMP. Significant post hoc comparisons were generally performed by comparing chi-squared values with critical values at the $\alpha = .01$ level.

			# reports commented on			# comments			# reports reported		
group	definition	size	min-mdn-max	total	% of total	min - mdn - max	total	% of total	min - mdn - max	total	% of total
CORE	release drivers, super reviewers, module owners, peers	928	1 - 68 - 40,017	226,697	46%	1 - 154 - 95,904	1,675,823	40%	0 - 16 - 2,809	59,035	12%
ACTIVE	assigned ≥ 1 report	2,568	1 - 25 - 24,191	435,764	88%	1 - 66 - 36,720	1,623,998	39%	0 - 9 - 5,145	168,644	34%
REPORTERS	submitted ≥ report	**119,707**	1 - 1 - 4,492	305,461	**61%**	1 - 2 - 6,802	806,979	19%	1 - 1 - 560	269,087	**54%**
USERS	all other commenters	29,674	1 - 1 - 1,057	27,249	5%	1 - 1 - 1,206	58,866	1%	0 - 0 - 0	0	0%
total		152,877	–	–		–	4,165,666		–	496,766	

Table 1. Definitions and aggregate statistics for each of the four contributor groups. In bold: reporters were the largest group, commenting on two thirds of reports and reporting more than half of reports.

CLASSIFYING BUG REPORT CONTRIBUTORS

Before assessing the user contributions to the Mozilla repository, we first needed to *define* users. We followed the "onion model" [4], which groups contributors into core and active developers, bug reporters, and passive users (among other more subtle groups outside of our scope). To identify contributors, we used the *email addresses* attached to reports and report comments, revealing 152,877 unique addresses (excluding nobody@mozilla.org). Some contributors used multiple addresses, evident from their reminders to use an alternate; other addresses may have represented multiple people. We did *not* try to merge addresses or differentiate comments from the same address, and so when we refer to "contributors," we are actually referring to email addresses.

To group these addresses, we used three measures. The first was whether a contributor had a *@formerly-netscape.com.tld* addresses or was listed on the Mozilla website as a member of one of the following groups: *release drivers* (who guide and manage fixes towards releases), *super reviewers* (who perform code reviews), *module owners* (who manage changes to "coherent bundles of source files"), *peers* (who help to *module owners*). This group, which we call **CORE** developers, included 928 email addresses. The next factor we used was whether the contributor had been *assigned* any reports, including 2,568 contributors. In Mozilla, policies state that the *assignee* of a bug should be the person "leading the effort to fix the bug" so we refer to these contributors as **ACTIVE** developers. The third factor we used was whether a contributor had reported at at least one bug; 119,707 of the remaining contributors satisfied this criteria, forming a group we will call **REPORTERS**. The remaining 29,674 contributors will be referred to as **USERS**, though we are careful to point out most of these contributors were probably power users like everyone else. **USER** contributions, by our definition, were limited to bug report comments and file attachments.

It should be noted that our grouping of contributors is based on a snapshot of the Mozilla website, and cumulative counts of report submissions and assignments. Therefore, each contributor was classified based on their *history* of contributions and not on their contributions at the time of reporting. How this affect our results is unknown; understanding contributions over time would be an useful direction for future work.

Descriptive and summary statistics for the groups are shown in Table 1. **REPORTERS**, by far the largest group, contributed 19% of comments, with most comments contributed by developers. The median **USER** and **REPORTER** *commented* on 1 report and the median **REPORTER** *reported* just 1 report; in fact, 64% of reporters only ever contributed to 1 bug. In comparison, the median **CORE** and **ACTIVE** developers reported and commented on an order of magnitude more.

To establish some convergent validity for our measures, we compared the *product* and *component* fields of the reports of each group. Groups differed significantly across both *product* (χ^2(df=135, n=496,750)=83,406, p<.0001) and *component* (χ^2 (df=1,827, n=496,750)=139,846, p<.0001). With respect to *product*, **REPORTERS** were more likely to report issues tagged *Firefox*, *SeaMonkey*, and *Thunderbird* (Mozilla's user facing products) than **CORE** or **ACTIVE** developers, while developers were significantly more likely to report on *Core*. With respect to *component*, the **REPORTER** reports were more likely to be tagged as *General, Tabbed Browsing, Plug-Ins, Bookmarks, Preferences* and *History*. These differences lend some convergent validity to our definitions.

In addition to this topical trends, it is also helpful to see associations between the four groups and their contributions over time. Figure 1 shows the number of contributors commenting every six months since Netscape released their source code in March 1998. Several things are evident from this graph. First, the **USER** and **REPORTER** groups are the only groups that fluctuate substantially in their contributions over time; the **CORE** and **ACTIVE** developers, in contrast, wrote a comparable number of comments each six months. Furthermore, as seen just before the release of Firefox 0.1, 1, 2 and 3, the **USER** and **REPORTER** groups grow until a major release, then drop off, then rise again before the next release. Because Mozilla software is updated regularly, this behavior is to be expected, with **REPORTERS** and **USERS** acting as beta testers, reporting on issues before each major release.

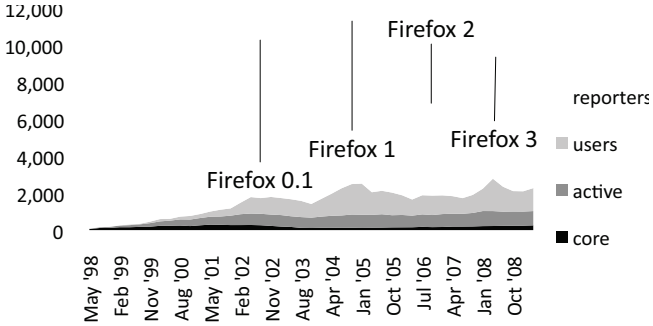

Figure 1. The number comment contributors, stacked by type, in each six month period since the release of the Netscape code. REPORTER and USER comments decrease after each release.

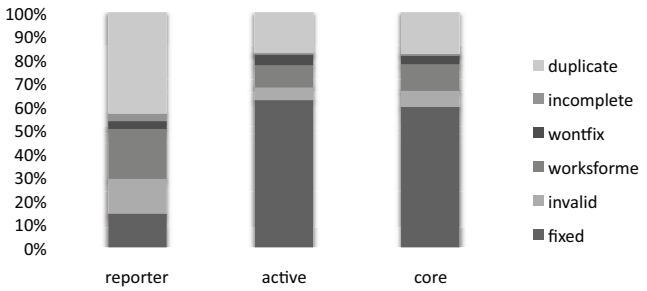

Figure 2. Resolution by reporter type. Few REPORTER reports are resolved as *fixed*; most are resolved as *duplicate*.

ANALYZING REPORT RESOLUTION

Having grouped contributors, we now move on to assess whether the contributions made by **USERS** and **REPORTERS** were of *value* to Mozilla. We operationalize *value* as the *resolution* field of bug reports with 8 levels. Mozilla policies state the meaning of each level as follows: *fixed* reports lead to a change in the software (a patch); *incomplete* reports are missing data needed to fix an issue; *invalid* reports identify a problem, but not one that Mozilla was responsible for fixing; *worksforme* reports did not involve a problem; *wontfix* reports identify issues that the community decided not to address; and *duplicate* reports regard issues that have already been reported. We omit the *expired* and *moved* resolutions from our analyses, since they were used infrequently.

As shown in Figure 2, there is a significant relationship between reporter *type* and the report *resolution* (χ^2(df=21, n=420,989)=117,303, p<.0001). About 62% of **ACTIVE** reports and 60% **CORE** reports are marked *fixed*, and these account for 79% of all *fixed* bugs. In contrast, 13% of **REPORTER** reports are marked *fixed*, accounting for 21% of *fixed* reports. About 40% of **REPORTER** reports are marked as *duplicates*, whereas the rest were mostly marked *worksforme* and *invalid*. Of the 119,707 **REPORTERS**, just 16,428 (14%) were responsible for the *fixed* reports. Of these, 65% had reported 2 or more reports, showing that experience was related to successfully reports.

How do **REPORTER**s' *fixed* reports differ from developers'? The *product* fields of **REPORTER** reports were significantly more likely to be *Tech Evangelism*, *Firefox*, *SeaMonkey*, and *Thunderbird* χ^2(df=135, n=148,902)=15,854, p < .0001. Moreover, the *severity* flag of **REPORTER** reports (set by reporters at reporting time), was significantly more likely to be marked *critical*, *major*, or *enhancement* and not *normal* (χ^2(df=18, n=148,902)=4,477,p<.0001).

Fixed **REPORTER** reports were also *open* significantly longer (measured from the date of creation to the date of closing) than **ACTIVE** and **CORE** reports (RS χ^2(df=3,n=148,902) =15,854,p<.0001). The median *fixed* **REPORTER** report was open for 371 days whereas the median *fixed* **ACTIVE** report was open for 123 and **CORE** was 119. This difference was *not* due to a delay in response: the median number of hours before the first developer reply was median of 5 hours across all divisions by reporter type and resolution. However, the days between the first patch being posted (defined later in the attachments section) and the bug being closed was significantly longer for **REPORTER** reports (RS χ^2(df=1, n=84253)=1046,p<.0001).

Reporter Duplicates were Mostly Redundant

What were the outcomes of the 96,219 *duplicate* **REPORTER** reports (which accounted for 40% of their reports and 22% of all reports)? For each duplicate, we checked the *resolution* of the report that the duplicate *referred* to (checking its dup_id field). When these pointed to other duplicates, our scripts recursively followed duplicate pointers until finding a non-duplicate report (6 reports were cyclic and were excluded from our analyses). Overall, 77% of *duplicates* referred to *fixed* reports and 73% of **REPORTER** *duplicates* referred to *fixed* reports. Duplicates by different reporter types referred to reports with significantly different resolutions (χ^2(df=21, n=102,355)=2300,p<.0001): **REPORTER** duplicates were more likely to refer to *worksforme*, *wontfix*, or *invalid* issues.

We can also consider the 44,819 *non*-duplicate reports to which duplicates *referred*. These were significantly more likely to get *fixed* than bugs without duplicates (χ^2(df=7, n=421,005)=13,449,p<.0001); one would expect widely reported issues to be addressed. However, when considering what *proportion* of these duplicates were reported by **REPORTERS**, the distribution is bimodal: in 52% of duplicated reports, **REPORTERS** were the *only* duplicate reporters; in 30%, there were *no* **REPORTER** duplicates; the remaining 18% had both **REPORTER** and developer duplicates. There is also a significant relationship between the *resolution* of duplicated reports and the proportion of **REPORTER** duplicates: (MLR χ^2(df=6,n=35,656)=2805,p<. 0001): as the proportion of reporter *duplicates* increases, so does the likelihood of it being marked *worksforme* or *invalid*.

To whose reports did **REPORTER** duplicates point? Of the 96,219 **REPORTER** duplicates, 56% were directed at other **REPORTER** reports, 65% of which were *fixed*. The other 44% of **REPORTER** duplicates were directed at **ACTIVE** and **CORE** reports, 84% of which were *fixed*. In these latter cases, it was possible that **REPORTERS** were actually first to report, but their reports were labeled *duplicate* anyway. In comparing **REPORTER** report creation times to the reports they referred to, this was true for only 8% of **REPORTER** duplicates.

What was the time frame in which **REPORTERS** reported duplicates, relative to the reports to which they referred? Across all **REPORTER** duplicates, 5% reported on the same day, 15% were reported between a day and 1 week after, 5% were reported within a month, 40% were reported between a a month and a year, and the remaining 30% were reported more than a year after. This does not necessarily mean that users were not contributing new information, since the median number of days a report with *duplicates* was open was 760 days. However, we used the report attachment data described in the next section and found that of the 55% of **REPORTER** duplicates that pointed to reports with patches, 66% were created *after* developers attached the first patch (and more were likely written after the problem had been diagnosed). In other words, most reporter duplicates were reported well after a draft patch had been authored.

Given these results, what value did **REPORTERS** contribute through duplicates? A small proportion were the first to report an issue, but the vast majority were reporting on issues that were already known and already being fixed.

group	# in group	# reports attached to	% of reports attached to marked *fixed*	# attachments	patches	plain text	logs/stacks	test cases	images	html	other	% replied to by devs
CORE	568	68,822	76%	151,680	51%	23%	4%	12%	7%	1%	2%	51%
ACTIVE	2,237	75,974	66%	170,290	47%	15%	5%	14%	9%	2%	9%	47%
REPORTERS	**19,071**	38,846	**22%**	62,868	9%	10%	9%	18%	**36%**	8%	10%	4%
USERS	2,584	2,618	32%	4,212	16%	16%	7%	15%	26%	9%	11%	12%
total	24,460			389,050								

Table 2. Attachment types by contributor groups and the proportion replied to by developers. In bold: most contributors who added attachments were REPORTERS who tended to attach images to reports that were not marked *fixed*.

Reporters' Mostly Attached Screenshots

Another form of contribution was *attachments* to bug reports. These included logs, images, error messages, code patches, test cases, and other information intended to facilitate problem diagnosis. As shown in Table 2, there were 389,059 attachments across 162,228 reports, by 24,462 unique contributors, most of whom were **REPORTERS**. Overall, 8,723, or 6% of *fixed* bugs contained **REPORTER** attachments.

To analyze these, we converted the MIME type of the file and the attachment description into one of *patch*, *plain* (a file of unspecified type), *test*, *image*, *html*, *log*, *stack*, or *other*. The most common **REPORTER** attachments were *images* and *test cases*, where as most developer attachments were *patches*. Of all reports with patches, 23% of **REPORTER** attachments were added *before* the first patch, 50% were added on the same day, and 27% were added after.

The resolution of reports was significantly related to the number of **REPORTER**' attachments on a report (MLR χ^2 (df=7,n=421,005)=713, p<.0001). As the number of **REPORTER** attachments increases, the likelihood of the report being marked *worksforme* increases as the likelihood of *fixed* decreases. In other words, a predominance of reporter attachments was associated with non-issues.

Reporter Contributions are Less Frequent, Less Useful

When **REPORTERs**' report resolutions are shown over time, as in Figures 3–5, we see a more nuanced view of their contributions. In Figure 3, we see that the *number* of **REPORTER** reports has been dropping since the 0.1 release of Firefox, and the number of *fixed* reports has dropped with it. In Figure 4, we see that the proportion of **REPORTERs**' report resolutions have stabilized, except for an increase in *invalid* and *incomplete* reports after the release of Firefox 1.0. In Figure 5, we see that the proportion of *fixed* reports due to **REPORTERS** reached its peak with the release of Firefox 1.0, and has dropped since. In fact, of **REPORTERS** *fixed* reports, 69% were fixed *before* the Firefox 1 release.

In comparing the topic of reports, the *product* fields of reports created before and after the Firefox 1.0 release were significantly different (χ^2(df=45, n=496,766)=188,985, p < .0001): before, reports were more likely to be about *Core* and after, reports were more likely to be about *Firefox*, *Thunderbird*, and *mozilla.org*. The same was true for the *component* field of reports (χ^2(df=609, n=496,766)=195,002, p < .0001): reports after the initial release were more likely to be about *General* or user interface components.

There are a several of interpretations of these trends. For example, perhaps most **REPORTER** effort before the release of Firefox 1.0 was from technically skilled **REPORTERS**, enthusiastic about the first release of the browser, but after this, less technically skilled **REPORTERS** dominated the reporting class, leading to more *incomplete* and *invalid* reports and fewer *fixed* reports. It is also possible that those **REPORTER** reports that would have been marked *fixed* began being marked *duplicate* instead, as **ACTIVE** developers became better at finding and reporting problems before **REPORTERS** reported them. Another interpretation is that as Mozilla software improved, there were fewer issues to report, and those issues that were reported were simply more trivial than the issues before the Firefox 1.0 release (perhaps because core Mozilla developers were focusing on repairing problems with the existing design, rather than evolving the design to satisfy new requirements). Whatever the case, one thing is clear: most of the valued **REPORTER** contributions occurred before the release of Firefox 1.0.

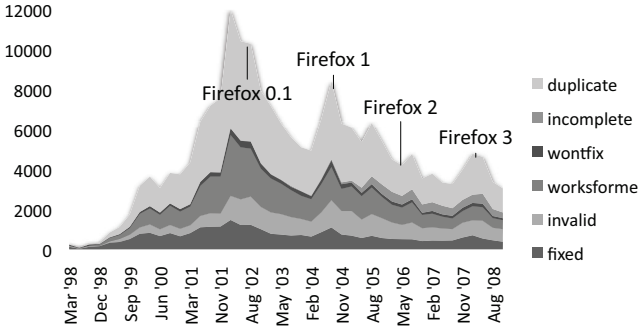

Figure 3. # of REPORTER reports by resolution per 9 months.

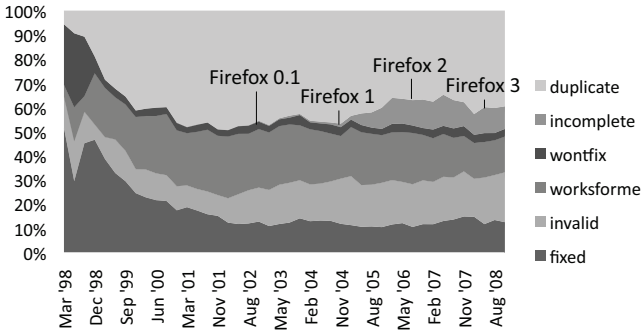

Figure 4. % of REPORTER report resolutions per 9 months.

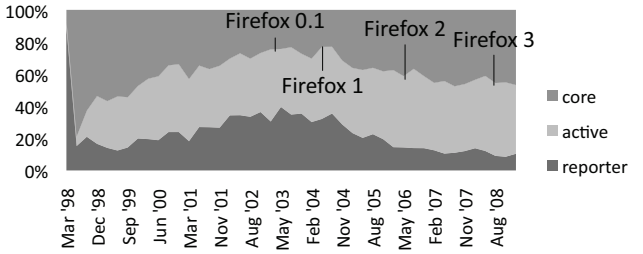

Figure 5. % of *fixed* reports by reporter type per 9 months.

group	% of reports	% fixed	[min – median – max] # commenters	[min – median – max] # comments
active + **core**	40%	62%	[1 – 3 – 41]	[1 – 6 – 700]
active + **core** + **reporters**	52%	17%	[1 – 3 – 65]	[1 – 4 – 315]
including **users**	5%	30%	[2 – 6 – 228]	[2 – 10 – 733]
only **reporters**	3%	7%	[1 – 1 – 8]	[1 – 2 – 23]

Table 3. Attachment types by contributor groups and the proportion replied to by developers.

ANALYZING REPORT COMMENTS

Thus far in our assessment of **REPORTER** contributions, we have found that most **REPORTER** reports were not *fixed*, that those that were fixed were reported by **REPORTERS** with substantial reporting experience before Firefox 1.0, and that **REPORTER** duplicates are usually reported after the issue was known and in repair. This does not mean, however, that less experienced **REPORTERS** did not make valuable, but more *subjective* contributions to report resolution through the report *comments* used to coordinate the resolution of a bug reoprt. For example, **REPORTER** duplicates may have helped developers identify other cases in which a problem occurs, or **REPORTER** and **USER** may have helped diagnose issues or offered user-centered solutions to design problems.

To investigate these possibilities, in this section we inspect small samples of representative reports with **USER** and **REPORTER** comments. To inform what kinds of reports to sample, we analyzed who was involved in report discussions by tabulating all reports against whether they involved each of our four contributor types. This revealed the four clusters shown in Table 3: those with *only* **ACTIVE** and **CORE** comments (40%), those with **REPORTER**, **ACTIVE**, and **CORE** comments (52%), those including all types, including **USERS** (5%), and those with *only* **REPORTER** comments (3%).

Of these, we were not concerned with the developer only reports, since those have been studied elsewhere [1,13]. Moreover, nearly all of the 3% of **REPORTER** only reports were marked *duplicate*, *worksforme*, and *invalid*, so we decided to pool these with other reports with these resolutions. However, 5% of reports with **USER** participation were different from other reports, as they were significantly more likely to be marked *worksforme* and *invalid* χ^2(df=7, n=421,005)=2619, p<.0001) and they had significantly more comments (RS χ^2(df=1,n=496766)=16206,p<.0001) and commenters (RS χ^2(df=1,n=496766)=27405,p<.0001). These 5% of reports alone contained 12% of all reports' comments.

Given these results, we decided to divide our analyses into two sets: (1) *routine* reports involving **REPORTERS** and any combination of developers and (2) *contentious* reports, involving **USER** comments. Furthermore, we divided our analyses of *routine* reports by resolution (again omitting *expired* and *moved* reports because of their infrequency).

What follows is an inspection of **REPORTER** and **USER** comments in these various reports types, based on random samples of 100 reports (and 40 contentious reports). Quotes include citations in form of (bug id: comment #: contributor type). We also report the number of reports that followed a particular pattern, but since we did not assess inter-rater reliability, these are only rough estimates of proportion.

Comments in *Fixed* Reports

The interactions between **REPORTERS** and developers in *fixed* reports (13% of **REPORTER** reports) were terse, highly productive, and largely concerned with repairing behaviors that both **REPORTERS** and developers believed were unintended. This is unsurprising, since as we reported earlier, the reporting experience of **REPORTERS** with fixed *reports* was significantly higher than other **REPORTERS**.

Of the 100 in our *fixed* sample, 43 involved a single **REPORTER** problem description, and a small number of status updates as developers wrote, attached, and reviewed a patch. In another 23, the **REPORTER** collaborated with developers to diagnose the problem by attaching logs, test cases, and screen shots, and then developers wrote a patch. In another 11, the **REPORTER** helped update and manage status flags and mark duplicates for an existing report. The 13 in which **REPORTERS** wrote "me too" comments contained valuable information that developers asked about and used to diagnose problems. In one case, a reporter even wrote a 1 line patch and asked to have it checked in. Only in 9 reports were **REPORTERS**' contributions potentially burdensome: in 6, a **REPORTER** proposed an idea that was turned down; in the remaining 3, **REPORTERS** asked a question about the bug resolution process. In general, the interactions in *fixed* reports were marked by a high degree of shared understanding about the *process* of bug fixing and the constraints on solutions.

Comments in Wontfix Reports

In contrast to *fixed* reports, most *wontfix* reports (3% of **REPORTER** reports) were requests for some new narrow expert feature of the form "it would be nice if I could..." Of the 100 in our sample, 53 were requests that were denied by developers because they were not broadly useful to "regular" users. In these cases, developers recommended writing a plug-in or add-on. In another 9, developers explained to the reporter how they could achieve the desired behavior with an existing plug-in or feature, essentially offering technical support. In one case, a **REPORTER** even proposed that the wording of the "about" dialog violated his religious beliefs:

> I have religious issues regarding the "I'll be careful, I promise!" button in the about:config nag screen. I believe promising is either the same as swearing. (503111:0:reporter)

In 19 reports, developers explained that the **REPORTER**'s request was moot, since they were no longer working on the software discussed. In 7, developers explained that the behavior reported was *intended* and would not change. In the remaining 12 reports, **REPORTERS** joined an existing report, indicating that they also wanted the feature, often offering design ideas about how it should work and providing use cases in which it would be useful. When developers replied to these, they usually identified reasons why the ideas would not improve user experience or why they were technically infeasible. Some **REPORTERS** expressed frustration:

> If you don't change Thunderbird, then Firefox on Mac must be changed, it must be done the same way. (383036:3:reporter)

> No, it must not. You don't get to make that decision, Firefox doesn't get to make that decision for Thunderbird … (383036:4:core)

Overall, it seemed that *wontfix* reports were generally written by power users with requests for narrow use cases.

Comments in *Incomplete, Invalid, Worksforme* Reports

In contrast to *fixed* and *wontfix* reports, *incomplete*, *invalid*, and *worksforme* reports (38% of **REPORTER** reports) were generally characterized by **REPORTERS**' lack of effort or skill in diagnosing problems before reporting them. Although we sampled 100 of each of these three resolutions, there was enough commonality in the nature of **REPORTERS**' contributions that we discuss them together (citing percentages instead of counts).

The most common kind of report (33%) involved a **REPORTER** identifying an issue that was already resolved in the most recent build. Developers' comments in these reports were of the form "have you tried the latest nightly build?" In most cases, the **REPORTER** did not reply and the developer closed the report. In a similar pattern (10%), developers asked the **REPORTER** if they had tried a particular remedy, and never received a reply (suggesting that the remedy worked). In some cases, the **REPORTER** replied to these suggestions and apologized for not trying them earlier. In another common pattern (24%), developers asked the **REPORTER** to provide more information to help understand the problem, but never received a reply. In a few cases, reporters did reply, but were unable to provide the information requested because they did not know how to use the diagnostic tools recommended by the developer.

The above reports, accounting for 67% of the *incomplete*, *invalid*, and *worksforme* reports we sampled, essentially constituted technical support. However, in 19% of reports, **REPORTERS** described unexpected behaviors that developers could not diagnose immediately. In these cases, developers continued to ask for information from **REPORTERS** before eventually discovering that the problems were due to exotic customizations that the reporters had made. Most were grateful to have their issue resolved:

> so i trashed the preferences and all was fine again. thank you all for your time. everyday mozilla is getting better, thank to people like you! (104347:7:reporter)

In 6%, the **REPORTER** identified an unexpected behavior that was ultimately correct, according to some specification. Developers usually referred to industry standards such as CSS, HTML, and JavaScript, explaining that other browsers did not properly implement the standard. Most of these were obscure edge cases where it was unclear to the reporter what the intended behavior was. Some, however, were closed *by design*, with developers offering some rationale for why the behavior was necessary or desirable. In another 5% of the reports, developers decided that the problem that **REPORTERS** identified was actually the fault of different system.

Interestingly, only 2% were deemed spam because they did report a problem, or cited some web site that did not exist. In general, most *incomplete*, *invalid*, and *worksforme* reports appeared to be technical support issues that were misdirected towards Bugzilla and should have instead been directed to *support.mozilla.org* or user forums. On the other hand, the ambiguity of many of these reports suggests that *whether* something is a bug is largely a matter of users' understanding of developer intent and responsibility, neither of which seemed to be commonly understood.

Comments in *Duplicate* Reports

Duplicate reports (42% of **REPORTER** reports), exhibited three basic patterns in our sample: 82 were marked as duplicate on the *same day* they were reported, with no additional comments (other than developer reminders to search for duplicates before posting). Another 12 involved some diagnosis about whether the report was a duplicate of something else, with the reporter often providing logs and other information to facilitate the diagnosis. The last 6 **REPORTER** contributions were "me too" comments, none of which were replied to or contained additional information.

It was possible, however, that the presence of duplicates alone was helpful. Therefore, across the whole data set we looked for comments with the word "duplicate" that were not automatically generated comments about duplicates, finding 44,093 uses in 32,092 reports. Of these, 35% were written **CORE** developers, 35% by **ACTIVE**, 27% by **REPORTERS**, and 2% by **USERS**. To inspect these in more detail, we sampled 100 of these comments. Of the 62 that were references to duplicate reports (and not a reference to some other concept of duplicate), 60% were developer statements that the report was a duplicate, 10% were statements that the report was *not* a duplicate, 8% were reporters apologizing for the duplicate, and 6% were developers telling people to stop filing duplicates. Additionally, 5% were statements about the poor quality of a report and 3% were reporters indicating that they could duplicate the problem on a certain platform. 5% were **REPORTERS** citing the number of duplicates to advocate for a change and 3% were **ACTIVE** developers referring to the duplicate count as an indicator of "unhappy users." In other words, few cases was the presence of duplicates used to advocate for a particular decision.

Comments in *Contentious* Reports

Reporter contributions to the reports in the previous sections seemed primarily to be technical support or spam. Did their contributions differ in the 5% of *contentious* reports, that we defined earlier, other than in having more commenters?

To begin, we classified the titles of the reports in our sample of 40, to get a sense of what issues were drawing so many participants. Topics included: bookmarks (12), the location bar (5), favicons (4), file type handling (4), keyboard shortcuts (4), installation (4), tabs (2), security (2), history (2), build configuration (2), and web forms (1). Moreover, 24 of the 40 requested a change in application behavior whereas the other 16 identified crashes, hangs, incorrect information or data loss. Clearly, contentious reports were marked by their relation to the design of major features of Firefox.

In our reading of these reports, **REPORTERS** and **USERS** contributions were many in number, but few in type, and markedly different than their contributions in routine reports. For example, reporters wrote "me too" comments, as they did in some *routine* reports, but here they read more like pleas, describing the dire implications of not fixing an issue:

> Forcing my users to retype the filename (presuming they even know what it should be) is **just plain oppressive**, IMHO... The organizations (large choral groups) for which I'm creating sites use the right-click save extensively to download (instead of play in their browsers) audio files for rehearsing music. ... It's just *code*, guys. Figure it out. (299372:49:reporter)

> I work as a sales support engineer for an "output management" software company... This bug or a related bug in Firefox has made it **completely incompatible** with our application. ... Over 50% of the web is delivering documents which are NOT html... If Mozilla dies it will very likely be due to this problem. There is no "happy middle-ground."(299372:100:reporter).

Developers responded to these comments by asking reporters to cease writing "me too" comments:

> To help this bug get fixed, the best thing would be finding someone to work on it. The next best thing would be not adding more "me too" comments -- **we know this is a problem**; it's just that there is no "easy fix." (299372:110:active).

Reporters often took these scoldings personally:

> Over two months ago I gave complete information on when and how I got the error AND spent a great deal of time isolating the messages that caused it ... Which part of that is just saying "me too"? **For crying out loud, I'm a nursing student, not a programmer**. Do you do your own x-rays before going to the doctor? (252697:10:user)

To reduce the frequency of "me too" comments, developers reminded reporters of Bugzilla's voting mechanism:

> Not to offend anyone new, but from here on out, can we refrain from spamming the bug with more comments along the lines of "it happens to me". **That is the purpose of the voting mechanism** in bugzilla. PLEASE read a bug fully before posting a new comment to it, and only then if there is information to add ... (242207:46:active)

Many **REPORTERS** and **USERS** did not heed these requests, likely because these were just one comment among hundreds. Others fashioned their own voting mechanisms out of the report meta data:

> Many apologies. I figured the blocking0.8 flag was meant as a vote sort of. BugZilla's confusing that way. And I figured I'd try setting it anyway, and if it was something I wasn't supposed to do, I wouldn't be allowed to do it. (210910:134:user)

Despite some developers' statements that "me too" comments were unhelpful, some **ACTIVE** developers did use them as evidence of the severity of a problem:

> i guess it would not be a big deal to make this much demanded feature (votes/comments/forum threads) available for Fx3.next and give it some adequate developer resources. (385077:21:active)

> The fact that this bug has been open for six years and only has 32 votes indicates that it's not a big issue. (171575:49:active)

In no reports in our sample did a **CORE** developer refer to comments in this way.

In addition to "me too" comments, **USERS** and **REPORTERS** wrote *design rationale* for the changes they desired. Most, however, were unsubstantiated generalizations about users:

> Joe user won't have the knowledge, nor the need to acquire it, and is unlikely to shoot himself in the foot using a feature he doesn't know exists. (385077:22:reporter)

> The average user will ignore the window, regardless if it said "YOU'RE BEING FOOLED" or not, they'll X out of the window. (337344:21:user)

> ... most users don't care about the Bookmarks toolbar (almost everyone I see has it empty, wasting space, not knowing how to remove it) (366797:256:user)

These generalizations often offended less technical users:

> Yet again, my behaviour as a user ... is apparently being second-guessed by some kind of would-be telepathy. Unless we actually have telepaths among us, could people please refrain from trying to use such totally unsubstantiated "top of the head" assertions about user behaviour ... (258301:25:reporter)

In contrast to these statements, **CORE** arguments were grounded in design questions and calls for evidence:

> Does the user expect the favicon to remain stored, or to be updated when the site icon changes? (240795:0:core)

> The real question is whether 3000ms of no feedback would cause the user to assume they somehow failed to click the right menu item and try the operation again. If it would, it's too long. The way to answer this question is through a usability study. (299372:232:core)

> As experiments, I think we should encourage people to try them and comment, but until we figure out whether or not they're solving real user problems, they're not suitable for inclusion into the mainline product. (366797:107:core)

Unsurprisingly, when **USERS** and **REPORTERS** did not see the change they desired, many were unhappy. Many expressed disillusionment, while also revealing their lack of understanding of the complexities of software development:

> Either Mozilla developers are too stupid to fix this kind of bug or they have a plot against their users. (242207:64:reporter)

> Why is it so hard for the programmers to get this right? It would be like me knowing the value of pi but not 1+1=2. (247884:88:user)

> Mozilla "Foundation", you have cash, you have the resources. FIX IT. PEOPLE DONATED MONEY TO HAVE YOU *FIX THIS KIND-OF SHIT*...This is the *EXACT* sort-of situation that shows why open-source *FAILS*. (299372:99:reporter)

If not ignoring these comments altogether, developers replied by citing the Bugzilla etiquette page and asking users to be more respectful. Some developers wrote comments to diffuse the situation by explaining the intent behind the process:

> OK, calm down everyone. How many times do I have to say it? It's what we want to try out to start with. That is not code for "we've made a decision" or "your arguments all suck and we're going to ignore them"... That's what the trunk is for. Experimentation. We want to do a UI experiment. (366797:217:core)

Not all users were disrespectful, however, even when the bug did not go in the way they desired:

> Thanks for the comment ... The technical reasons for why places.sqlite shouldn't be placed on a network share are a lot clearer, and I see why you wouldn't want to do that. (385077:40:reporter)

And when contentious bugs did get resolved in their favor, reporters were often quite grateful:

> Thank you for putting this back on the TODO list, Chris. ... small polish like this does add up to something, especially for IE users that would consider switching browsers. I would do it myself but I don't know how to code. (195031:26:reporter)

In summary, **USER** and **REPORTER** contributions to contentious reports were dominated by misunderstandings: users did not understand the bug resolution process or the technical difficulty of devising viable solutions. Users were also quite egocentric in stating their concerns, many of them suggesting they were entitled to a prompt fix. Developers' comments, in contrast, focused largely on trying to explain to the confluence of commenters what was happening, what was constraining the design, and why such constraints were inevitable.

DISCUSSION

Throughout our analyses, one basic trend has emerged: the majority of **REPORTER** reports did not lead to changes, did not appear to contain valuable information, and in most cases, devolved into technical support or were simply spam. Those **REPORTER** reports that did lead to change were largely reported by a comparatively small group of about 8,000 experienced, frequent **REPORTERS** before the release of Firefox 1.0. Moreover, when inexperienced **REPORTERS** *did* contribute reports that led to change, they were open far longer (a median 6 months longer) than other fixed reports, raising the question of whether these issues were less critical.

In our discussion, we consider various implications of these results from testing, HCI, and design perspectives.

The Software Testing Perspective

One of the central claims behind open bug reporting is that the work of software testing can be delegated to users. This did hold true, but doing so appeared to entail many costs. For example, we found that most reporter efforts were at reporting critical issues of which **CORE** and **ACTIVE** developers were already aware. Moreover, **REPORTER** reports that *did* identify unique issues may have been less critical (based on the longer period that they were open). Moreover, the claim that "many eyes make all bugs shallow" [12] did not hold for Mozilla bug reporting: the median report had only 2 contributors, most of whom were **CORE** and **ACTIVE** developers. It would be more accurate to say that many eyes led to a few high quality reports. In some ways, the roughly 8,000 experienced reporters who reported fixed bugs do not differ from the 1000's of beta testers recruited by closed source companies.

Of course, this perspective ignores the possibility that *without* open bug reporting, Mozilla may never have recruited its thousands of **CORE** and **ACTIVE** developers. In fact, a large amount of work demonstrates that bug reporting is a primary entry point for newcomers [5,7,8,9]. Perhaps the value of open bug reporting is more in recruiting, vetting, and retaining talented developers, and that unwanted content is the *cost* of this recruiting. And these costs appear to be insignificant. Mozilla has handled approximately 270,000 reports from **REPORTERS** over the past 11 years, which is an average of 67 **REPORTER** reports per day, spread across 3,500 **ACTIVE** and **CORE** developers. This is only an average of 1 **REPORTER** report per developer every 50 days. Even though only 1 in 6 of these ends up being valuable, this is a very small productivity loss. Moreover, this delegation approach scales, because for every ten new reporters, there is one reputable developer to evaluate their contributions and decide, by merit, whose contributions are of value. In this sense, open bug reporting is like an extended job interview.

How might these findings translate into *closed-source* software development contexts? If the primary benefit of open reporting is in recruiting *developers*, it appears that closed-source companies have little to gain from open repositories. This is because what appeared to make *experienced* reporters effective was their understanding of what constituted an bug and what was by design. Volunteer reporters with no knowledge of the *intended* design are perhaps more likely to report on issues that were intended. In essence, an open repository with closed source (which some projects have adopted, including the Facebook API), would likely begin to look more like traditional technical support.

The Human-Computer Interaction Perspective

Another party in this discussion are the users themselves. While users may have received many benefits from reporting, such as technical support and the rare fix, their experiences were not altogether positive. When users tried to be helpful by contributing as much data as they could, they were called spammers and told to stop. When developers tried to explain the problem trying to be solved, users interpreted the technical complexity as excuses for not implementing a change. What underlies these user misunderstandings seemed to be a legitimate confusion about developer intent and responsibility. How were users to know what behavior the developers intended, other than reading potentially out of date specifications and project roadmaps? How were users to know whether a bug was Mozilla's responsibility or some other organization's? Many reporters only knew that their web site did not work; they did not know that this was because the web site had a bug, or was designed for the quirks of Internet Explorer. These misunderstandings illustrate many kinds of unhelpful culture clash, which only appeared to annoy developers and embarrass users.

The Design Perspective

The discussions above raise several issues for the design of open bug reporting tools. For example, one critical question is whether the *openness* of Bugzilla was actually helpful. Our results show that one of the most common problems was that users did not perform basic diagnostic steps before reporting a problem, despite a number of support resources such as dozens of Mozilla user forums and *support.mozilla.org*. In these, reporters probably would have resolved their issue without taking valuable time from developers. Instead of allowing *anyone* to report issues, perhaps a better strategy is to delegate reporting to a trusted group of reporters, and have them receive issues through support sites first. This would add a new class of quality assurance gatekeepers (as is already done in commercial support), while preserving the ability for new contributors to acquire status and reputation.

Other design issues have more to do with reporting tools and groupware. For example, our results show that while *duplicates* are many in number, the challenge was not necessarily in *detecting* duplicates (since most duplicates were quickly marked so after the duplicate was posted), but in aggregating the information they contained and making it clear what was already known. Reporting tools like Bugzilla should provide ways to organize information provided by reporters, and incentivize reporters to add it there rather than as comments. Even something as simple as a wiki at the top of a report for users to indicate the platforms that exhibit the issue, the context in which it occurs, and ideas for fixing it, could go a long way in preventing unhelpful "me too" comments by making it obvious what has already been said.

Another common problem was that **USERS** and **REPORTERS** simply lacked any knowledge of the Mozilla bug reporting process and the developers involved in a particular report.

Adding some meta data, and perhaps structure, to reports, could remedy many of the misunderstandings that these knowledge gaps led to. For example, all reports move through a set of states; bug reports should clearly show these states, to make it clear that developers have moved on to fixing a problem (and so no more comments that the problem exist are needed). Providing information about the developers involved in the report, such as how many other bugs they are assigned, and a schedule of the upcoming releases toward which developers are working, would allow reporters to have more patience and understanding when making requests.

Aside from the problems we observed, there are several *opportunities* that our results revealed that could both broaden participation in open bug reporting tools while minimizing unwanted content. For example, many of the *wontfix* bugs were simply usability problems in specific situations that the developers viewed as uncommon. If feedback and feature request mechanisms were built into the software itself (for example, imagine clicking on a UI control and typing, "I always accidentally click this"), the Mozilla community could *automatically* gather evidence about set of the situations that users are experiencing frustration. Not only would this be a great source of usability information, but if a developer eventually writes a report about the problem or feature request, there would be data to assess to what extent the situation occurs, helping to prioritize issues.

Threats to Validity

Our study has a number of limitations that limit its generalizability. First and foremost is the fact that we only studied the Mozilla project, which is someone unique in its corporate origins and user-facing nature. Many open source projects are aimed at more technically savvy users, and the quality of reports in these developer-centered projects may differ. Moreover, Mozilla and its user base is large compared to other open source software, which may be a central reason for the results we observed. Also, bug reports are not the only venue through which users contribute: users may add value through mailing lists, IRC, and other channels.

In additional to these generalizability concerns, there are a number of internal validity concerns. Furthermore, the substantial scripting involved in deriving our results may have been effected by scripting errors in unpredictable ways. Our classification of contributors was *static*, and did not account for changes in status over time. This may have led many of the contributors who would have previously been grouped as **CORE** to be included in other groups instead.

CONCLUSIONS

In this paper, we investigated to what extent power users provide valuable contributions in open bug reporting. Our study found that in the case of Mozilla, they primarily did not, but what Mozilla gained was a small pool of talented developers and a number of critical fixes before the release of Firefox 1.0. While the amount of unwanted content was high, the cost of filtering these reports was spread over many developers. Our results suggest many ways that open bug reporting tools could be improved to reduce unwanted content and maximize the value that both users and developers get out of open bug reporting.

ACKNOWLEDGEMENTS

We thank Alex Faaborg of the Mozilla Corporation for his insights on Mozilla bug resolution.

REFERENCES

1. Barcellini, F., Detienne, F., Burkhardt, J.M., Sack, W. 2008. A socio-cognitive analysis of online design discussions in an open source software community. *Interacting with Computers,* 20:141-165.
2. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R. and Zimmermann, T. (2008). What makes a good bug report? *ACM Foundations of Soft. Engineering*,308-318.
3. Cooper, A., Reimann, R. and Cronin, D. 2007. *About face 3: The essentials of interaction design*. Wiley Pub.
4. Crowston, K., Annabi, H., Howison, J., and Masango, C. 2004. Effective work practices for software engineering: free/libre open source software development. *Workshop on Interdisciplinary Software Engineering Research*.
5. Ducheneaut, N. 2005. Socialization in an open source software community: a socio-technical analysis. *Computer Supported Cooperative Work* 14(4):323-368.
6. Hendry, D. G. 2008. Public participation in proprietary software development through user roles and discourse. *Int'l J. of Human-Computer Studies*, 66(7): 545-557.
7. Herraiz, I., Robles, G., Amor, J. J., Romera, T., and González Barahona, J. M. 2006. The processes of joining in global distributed software projects. *Global Software Development*, Shanghai, China, 27-33.
8. Jensen, C. and Scacchi, W. 2007. Role migration and advancement processes in OSSD projects: a comparative case study. *International Conference on Software Engineering*, 364-374.
9. von Krogh, G., Spaeth, S., and Lakhani, K. R. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32 (7):1217-1241.
10. Li, Q., Heckman, R., Allen, E., Crowston, K., Eseryel, U., Howison, J., and Wiggins, A. 2008. Asynchronous decision-making in distributed teams. *Computer Supported Cooperative Work*, 1-2.
11. Mockus, A., Fielding, R. T., and Herbsleb, J. D. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. on Software Engineering and Methodology*, 11(3):309-346.
12. Raymond, E. 1999. *The Cathedral and the Bazaar*. O'Reilly, Sebastopol.
13. Sandusky, R. J. and Gasser, L. 2005. Negotiation and the coordination of information and activity in distributed software problem management. *ACM Conference on Supporting Group Work*, 187-196.
14. Singh, V. and Twidale, M. B. 2008. The confusion of crowds: non-dyadic help interactions. *ACM Conference on Computer Supported Cooperative Work*, 699-702.
15. Yamauchi, Y., Yokozawa, M., Shinohara, T., and Ishida, T. 2000. Collaboration with Lean Media: how open-source software succeeds. *ACM Conference on Computer Supported Cooperative Work*, 329-338.