

Users vs. Developers: Defect Detection and Resolution in Mozilla's Firefox

Jean-Michel Dalle
Université Pierre et Marie Curie
(Paris 6)

Matthijs den Besten
Oxford e-Research Centre
University of Oxford

Laetitia Canini
Université Pierre et Marie Curie
(Paris 6)

Draft: March 7, 2009

Vulnerability discovery, disclosure, and patching are part of the more general process of defect detection and resolution that defines the maintenance of software systems. On the basis of a large collection of bug reports retrieved from the Bugzilla issue tracking system for Mozilla's Firefox Internet browser, we have started to identify the tensions and strains that appear from this daily interactions in this emblematic open source project. In particular, we find preliminary indications pointing to a tendency to ignore contributions from outsiders who have not yet built a reputation through earlier interactions. Our research is still ongoing, but what this suggests already is a limit to the absorptive capacity of open source software communities in general that might be reason for worry in particular in those cases where highly critical software is maintained by only a small number of people.

1 Introduction

An important aspect of the economic literature about open-source software deals with the involvement of users in development processes. For instance, in a study of Apache's security software, Franke and von Hippel (2003) argue that Apache is able to accommodate heterogeneous user needs because it is structured as an highly configurable "innovation toolkit." The highly modular structure that is typical for this kind of software would allow users to recombine as they see fit and to "develop exactly what they want, rather than relying on manufacturers to act as their (often very imperfect) agents" (von Hippel 2005).

An opposite argument, often heard in the early days of the emergence of the open-source phenomenon, stressed on the contrary that developers would be attracted to ‘sexy’ work and would therefore tend to neglect more mundane tasks — they would rather contribute to the Linux kernel rather than to the driver for an obscure printer. This approach stresses the role of reputation within open-source communities, the heterogeneity of motivations, and so on (Dalle and David 2007).

Although we are convinced that both of these theoretical approaches convey important elements of truth, and that they apply jointly in many cases — for instance, the first to explain participation in open-source communities, together with other more extrinsic motives (Lerner and Tirole 2002), and the second to analyze the allocation of efforts within open-source communities, and therefore their organizational processes — we suggest that further inquiries are still needed to clarify the respective role of users and developers within open-source and related communities. Indeed, analytical work by economists (e.g., Bessen (2002) and Arora, Caulkins, and Telang (2006)), and technical analysis by software engineers (e.g., Banker and Slaughter (1997)) has revealed reasons why, in the case of computer code, it will be optimal for producers to allocate resources to extensive post-release repair of defects based on feedback from user-experience. To put it differently, and somewhat bluntly: the economics of the software industry is affected by the shift of a good part of the development costs and of the development burden to the “post-release” stage and this is not without consequences for the structure of the markets, nor for the quality of the products. To put it in yet another way, software goods are dynamic goods released in sequential versions, and economic theory is still relatively far from being able to analyze these characteristics in appropriate depth.

Understanding the interaction, and possibly incentive issues, between users and developers throughout the process of software development would be an important step forward. As a matter of fact, such inquiries have become more feasible in the case of open-source software because of the availability of development traces both in repositories of the software source code, the code bases, and in databases of fault reports, the bug repositories (see below). This allows us to track and analyze development traces for evidence of “users vs. developers” interactions, and possibly misaligned incentives. We do this in the case of the Mozilla project, and specifically of the Firefox browser, less because of its large distribution than because of its dedication, from scratch, towards nave users. Another important, pragmatic, reason for our choice is that the Mozilla community has also given birth to the issue (bug) tracker Bugzilla and makes extensive use of it. This allows us to access considerable amounts of development data besides code traces, pre- and post-release. Note that we feel that a comparison between open source software development and proprietary software development would be more than appropriate. At the moment, we focus on open-source software only because we lack sufficient data with respect to proprietary software development.

Section 2 and 3 introduces several relevant contextual elements about Mozilla / Firefox, and Bugzilla, respectively; section 4 suggests anecdotal evidence of misaligned incentives between users and developers taken from 3 specific bug reports; section 5 then presents quantitative results obtained with survival analysis techniques that stress differences between pre- and post-release development processes, on the one hand, and

provide further support to the existence of post-release misaligned incentives between users and developers, on the other; section 6 concludes.

2 The Mozilla project and the Firefox browser: dedicated to “Mom & Dad”

When it was founded in 1998 in an attempt to save what was left from Netscape after it had lost the so-called “browser wars” against Microsoft’s Internet Explorer, Mozilla faced several challenges: On the one hand, it wanted to encourage user-innovation in order to benefit from a larger user-base and a faster developing product, but at the same time it needed to find a way to channel these innovations back into its ongoing browser development. In order to encourage user-innovation, Mozilla carried out a major overhaul of the code-base that it had inherited from Netscape and transformed this piece of software into a more modular toolkit that was easier to understand and modify by outsiders (MacCormack, Rusnak, and Baldwin 2006). In addition, Mozilla put in place a system for change management in order to track the treatment of bug reports and feature requests. In the terms of Athey and Schmutzler (1995), Mozilla sought to combine product flexibility with process flexibility in order to enhance demand and reduce costs.

In the first years that followed this open-sourcing of its software solution, it looked as if Mozilla would fail. At least until the restructuring of its code base mentioned above, Mozilla struggled to attract and retain volunteer developers to the project even despite the vocal support that Mozilla was given by leaders in the open source movement (e.g. Raymond (1999)). In 2002, four years after its foundation, many felt that Mozilla’s browser software had become even more bloated and harder to use than what Netscape had left behind. It was for these reasons that a small group of developers “forked” and started a separate project called Phoenix that would eventually evolve into Firefox. Blake Ross, one of the initiators, explained the rationale for this decision as follows (McHugh 2005):

The Mozilla product at the time had about 10,000 options. You basically needed to know the secret handshake to get anything done. It sounds corny, but it was important to make something that Mom and Dad could use.

This minor rebellion marked a turning point for Mozilla. Firefox was pieced together and started to flourish. Moreover, as Firefox flourished and regained market share from Microsoft’s Internet Explorer, the process that had created it became a model for participatory, open source collaboration (Mendonca and Sutton 2008).

Paradoxically, Firefox has several features that would seem to make it rather atypical as an example of open source collaboration. For example, one important feature of Firefox is its relative centralization of power in a small group of developers who make the ultimate development decisions. Centralization may have been necessary as some kind of hierarchy was needed to match problems with the existing browser with people who knew how to solve them (cf. Garicano (2000)). More in general, centralization

seems to be a way to transform user innovations into commercial products (Baldwin, Hienerth, and von Hippel 2006). Another feature of Firefox is the interest that its lead developers expressed for “Mom and Dad.” Rather than searching for some sort of cosy consensus among the community, lead developers at times chose to put themselves at odds with the community in order to defend the interests of those outside — a stance succinctly expressed in this message from Blake Ross (Livingston 2007):

We’re making a product for mom and dad. You still have a voice here, but some of the features that you think we should add may not be the ones that they want to use. So you have to take our word for it that, even though 500 of you want something right now, you may actually be in the minority of a much larger group that we’re pursuing that’s going to be silent during this phase of development.

As Firefox succeeded in attracting mom and dad, it has also become rather atypical as an example of open source for its ratio of users to developers and for the volume of change requests that has come from these users. Consequently the ability to filter among these requests and assign a priority to the most pertinent among them is probably even more important in case of Firefox than it is in general. Decisions about what priority to assign to bugs and feature requests are hard because it is not always clear how addressing the suggested changes will impact the product as a whole (cf. Villa (2003)). In most cases where the open source development model has been successful, the requirements of the developers coincide with those of the end-users (Kuan 2005). In the case of Firefox it was recognized that the requirements were different. This situation made decision about priority more difficult as it forced developers to take account of requirements and desires that were not well articulated. At the same time, lead developers of Firefox also faced pressure from their peers as is evident from this short extract from an interview with Blake Ross, who was one of them (Ryan 2006):

- Olivia Ryan: And so was it difficult sometimes to strike that balance between working on an open project and trying to keep end-users in mind?
- Blake Ross: Yes. Yes. Everybody hated us for a long time.

In a sense, Firefox’s lead developers found themselves in a situation not completely dissimilar from referees in a soccer game who have to decide whether injury time or not, and it is known that referees tend to yield to the home team under pressure (Garicano, Palacios-Huerta, and Prendergast 2005). And so, as it tried to satisfy heterogeneous needs, Mozilla carried out a delicate balancing act. As we shall now see, the system of change management, which is an important channel of communication between Mozilla’s lead developers and its end-users, played a crucial role in helping Mozilla achieve this balance in the case of Firefox.

3 Bug tracking in Mozilla and the CanConfirm privilege

Software, as noted before, is a dynamic good. Over the years a practice of software configuration management has emerged to deal with the near-continuous change that

is typical for software (Estublier et al. 2005). This practice encompasses a range of activities among which version control and issue tracking are the most common. Mozilla relied on a widely used tool called CVS for version control of its code base and it relied on Bugzilla to keep track of the issues collected in its bug repository. Bugzilla has been developed by the Mozilla community and has been adopted by a large number of open source projects since then (Halloran and Scherlis 2002). The Firefox subproject also used Bugzilla, but appears to have done in a manner that departed somewhat from previous practice as it tried to address the concerns of “mom and dad.” In particular, Blake Ross and other early leaders of the Firefox project had clear ideas about how the bugs in Bugzilla should be triaged: as mentioned before, they explicitly announced that they would ignore or downgrade some of the reports filed on Bugzilla because they feared that otherwise priorities would be skewed in the wrong direction as bugs would predominantly be filed by power-users who would suggest so-called “alpha-geek” features, much more than “mom and dad” would like.

Figure 1 is a schematic outline of the process of bug resolution that is adopted by projects that employ Bugzilla. When someone reports a bug, this bug will be labeled either “Unconfirmed” or (confirmed as) “New.” The criterion used to decide which label applies are within the control of the project, and in the context of the Mozilla project, to be specific, if the reporter was an outsider with little or no involvement in development the bug’s initial status would be “Unconfirmed”, whereas the label “New” would be affixed to the bug if it had been reported by an insider. Indeed, only bug-reporters who have the so-called CanConfirm privilege are able to report a bug as “New” or to turn an “Unconfirmed” bug into “New.” This privilege is granted by cooptation, based on the past track record of developers asking for it. We shall therefore consider for the sake of our analysis in this paper that the CanConfirm privilege characterizes members of the core of the community, as compared to more peripheral members without this privilege.

The distinction between core and periphery is a particularly important one to make in the context of open source software development. While the distinction between insiders and outsiders may be less clear cut in open source development communities than in traditional firms, some of it still lingers as typically a large part of the development of open source software is done by a relatively limited core of developers, helped by a larger periphery engaged in supporting activities ranging from the compilation of manuals to the detection of bugs (Lakhani 2005; den Besten, Dalle, and Galia 2008).

In the context of Firefox, it would seem very likely that one would therefore find users among the reporters of “Unconfirmed” bugs, especially among those bugs reported after the first stable release (Firefox 1.0). Before this first stable release, “Unconfirmed” bugs would more typically originate from developers who were not intimately involved in the development process, or in the process of joining it. As a consequence that will be useful below (section 5), the fact that some bugs start as “New” while others start as “Unconfirmed” gives us important information about the status of the bug-reporters.

Having entered the process, the bug is first *triaged*. That is, it is in particular assigned a priority, and is eventually assigned, i.e. a developer is in charge of dealing with the report. Next, the bug is *treated*. That is, the cause of the problem is identified and patches are proposed. Once a solution is accepted, the bug is declared “resolved” and

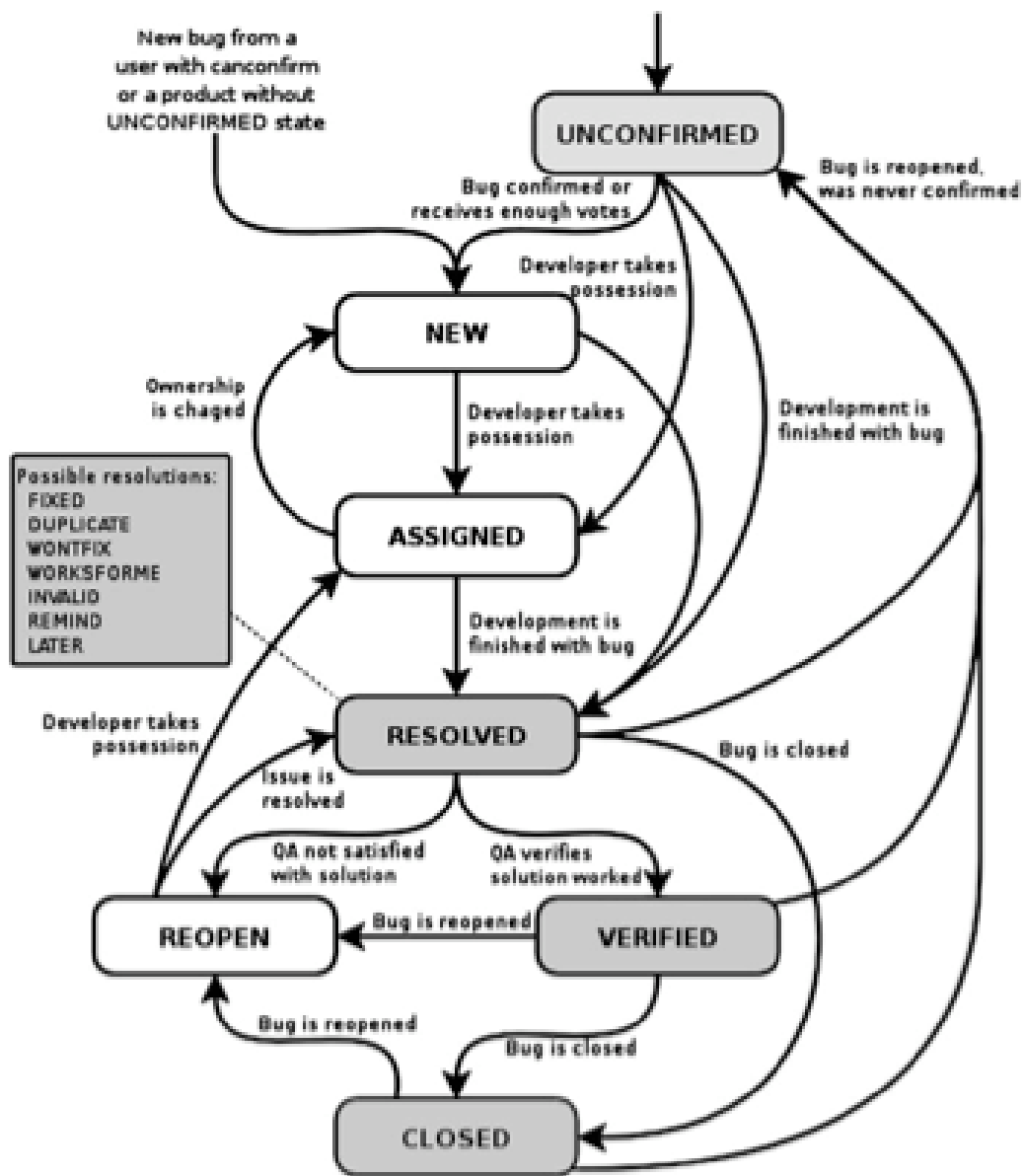


Figure 1: Schematic outline of the Bugzilla bug resolution process (source: The Bugzilla Guide)

after it has been verified that the solution (patch) works satisfactorily when integrated into the code base, the bug is declared “closed.”

Note however that there are a variety of other ways to close a bug apart from properly speaking resolving the problem it identifies. In consultation with other people tracking the bug, the developer who has taken possession can for instance decide that the problem identified does not exist (“works-for-me”) or, on the contrary, that it does exist but has been reported and is being treated elsewhere (“duplicate”). In the latter case, one of the two reports is closed as Duplicate, while a message is posted on the forum of the other indicating that Bug #XYZ was a duplicate of this one.

4 “Users vs. Developers”: anecdotal evidence from 3 bug reports

The tension between users and developers is palpable in many of the bug reports that are maintained in Bugzilla and a closer look at some of them can provide many insights in the multitude of ways in which this tension plays out.

Consider bug 213186.¹ This bug report is a request to alter the text in the preference pane of the browser menu and in replace the statement that “Cookies are delicious delicacies” with a more appropriate explanation of cookies, a technical term, for concerned users without prior knowledge of the concept. Blake Ross, the Firefox initiator, was responsible for the “delicious delicacies” joke himself in the first place and his mom and dad might well have approved. Many other users however might be put off by the lack of information provided to them if they seriously wanted to consider whether want to accept cookies or not. That is why this bug was marked as one that had to be fixed before the official release of Firefox 1.0. The record in Bugzilla shows that the phrase “delicious delicacies” was eventually replaced by “pieces of information stored by web pages on your computer[...].” However, considerable effort seems to have been required to resolve this bug as developers were determined to hang on to this bastion of insider humor against the wishes of people who yearned for something a bit more “professional.” It required over a year to resolve in which 18 persons made 55 comments and 9 possible solutions were proposed as patches. Many continued to prefer the quirky “delicious delicacies” definition over the more mundane one that replaced it and in true open source fashion an extension to Firefox was developed that restores the old description for Firefox 1.0 till 1.5.² In later versions, a definition of cookies is left out altogether.

The bug record for the “delicious delicacies” bug is an example of a mostly internal discussion among the software developers. Yet, there are also many bug records that show something quite different: the attempts of the external user-community to get heard. The record of bug 171349 which reports that the Windowing System’s Standard Icon appears as Firefox icon is such an attempt.³ This bug, important to be resolved in order to give Firefox a more professional and clearly recognizable appearance, took so

¹https://bugzilla.mozilla.org/show_bug.cgi?id=213186

²<https://www.squarefree.com/extensions/delicious-delicacies/>

³https://bugzilla.mozilla.org/show_bug.cgi?id=171349

long to be treated that peripheral bug contributors started to despair. As one commenter remarked:

This bug was first discovered about 2 1/2 years ago and is still listed as “NEW”?

Upon which someone else replied:

Mozilla, like all the other big projects, has a LOT of bureuacracy [sic] going on. I reported this so long ago I forgot all about it, and probably would've submitted it again if it weren't for the recent activity of my report being marked as a dupe... But even if it requires 5 people to check it and give it their OK, surely adding a .ico to the package can't be /that/ hard, can it?

In the end, it took 31 duplicate reports of this problem, 200 comments to which 89 people participated with 11 proposed solution to bring this bug to a close after almost three years. The reason that this particular bug took so long to close, was not that the core developers did not notice it. They also do not seem to have given the bug as serious a consideration as they would have in case they had known the bug-submitter. In comment 25, Blake Ross himself steps in and marks the bugs as a duplicate of another bug. An assessment that the original bug-submitter disputes while also making clear that he doesn't care anymore one way or another. A few comments later, another developer finally agrees with the original submitter and offers to work on the bug on condition that the submitter provide more contextual information. This the submitter cannot do, as it has already been such a long time ago since he came across the problem and hasn't used the software since. We then see responsibility for the bug being passed around from one developer to the other and it is only after the resolution of the bug has been linked to the target milestone of Firefox 1.0 that proposals for solutions start to come in. Were it not for the persistence of the original submitter of the bug report, this report would probably have remained classified as duplicate of another bug, as it had become straight after the report, and would not have been dealt with directly.

Sometimes, Bugzilla can also become a place for coordinated action. Bug 439604 is an example of this.⁴ It illustrates how people from the periphery can still use the Bugzilla change management system to get their point across. This bug is an expression of discontent about the fact that the latest version of Firefox requires the user to agree to an end user license agreement (EULA) upon installation. The bug has been submitted by someone from the periphery as is evidenced by the bug's initial status of “Unconfirmed.” Nevertheless the submitter and his allies are clearly at home in Bugzilla: the bug description and subsequent follow all the right conventions of proper bug reports. Moreover, the bug-reporter has been able to mobilize support and get people to vote on the importance of the bug classifying it as very important. In this case, the pressure thus generated seems to have worked: Mozilla relented and the EULA requirement was removed. However note that outrage was not only expressed through this official channel but also at public places like Slashdot. And other interesting detail to note is that this

⁴https://bugzilla.mozilla.org/show_bug.cgi?id=439604

is not the bug report that is acted upon according to the records. Instead, the record of bug-resolution can be found as bug 443918 — a bug that expressed the indignation of one of the core developers.⁵ Lerner and Tirole (2005) notice with respect to software licenses in general that more mainstream oriented projects tend to have different licenses from developer-oriented projects. Concern with the imposition of end-user license agreements seems more likely to stem from open source idealists than end-users who are accustomed to click through in any case. Hence, the resolution of bugs 443918 and 439604 was probably more about keeping developers on board than about making sure that the best was done for “mom and dad.”

5 “Users vs. Developers”: quantitative evidence

Methodology From the CVS, we retrieved for each bug the number of different authors referring to the bug in their commit-comments; the number of files touched by these commits; the number of distinct comments and the number of commit; and, finally, the number of lines of code added and deleted through the commits.⁶ From the bug-reports, we retrieved the number of persons copied in the bug resolution discussion; the number of other bugs the bug depends on or blocks; the numbers of attachments, patches and comments and the number of distinct contributors to the discussion; and, besides, the number of bugs that were identified as a duplicate of the bug reported. In addition, we retrieved the priority assigned and the severity estimated for each bug. With help of the change-log of the bug-reports, we also retrieved the time that passed between the opening of the report and the assignment of someone in charge of the resolution process; we retrieved the number of times the bug was (re-)assigned whether its priority was incremented or decremented or had been changed more than once; whether its severity increased or decreased or changed more than once — where the importance was judged to be in order of trivial, enhancement, minor, normal, major, critical, blocker. We determined whether the initial status of the bug report was New or Unconfirmed; whether the bug was reopened at least once; the number of report edits by the bug-reporter, and the number of report edits by the last person in charge of its resolution.

We drew two samples of bugs from the 40 000 or so that have resulted in a change to the Firefox code base. The first from the early stages of the project, when Firefox was still called Phoenix, in between release 0.1 and 0.5; the second from a later, more mature, phase between release 1.5 and release 2.0. For the moment, we ignore bugs where the severity was judged to be an enhancement as we want to focus on bug reports rather than feature requests.

Limitations Sack et al. (2006) note that developers coordinate their activity in three main information spaces: “the implementation space, the documentation space, and the discussion space.” Our collection covers each of these spaces, at least partially. It

⁵https://bugzilla.mozilla.org/show_bug.cgi?id=443918

⁶See Ayari et al. (2007) for a discussion of the pitfalls in trying to retrieve bug-numbers from CVS comments)

covers the implementation space in so far as we only consider bugs that are mentioned in commits to the official code base that were kept by the Mozilla concurrent versions system (CVS). For those bugs, in so far as they are associated with a file of which at least one revision is part of a Firefox release, we retrieve the bug-reports and the logs of changes to those bug reports kept by the bug tracker system Bugzilla. Thus, we are covering the documentation space as well. We also cover the discussion space since mailing-list style comments relating to the bugs are included in the reports. However, if Blake Ross is right in his assessment that Internet Relay Chat “was a big means of communication and especially with Firefox early on” (Ryan 2006), then we might have a problem, because, too our knowledge, archives of those chats have not been kept.

Analysis In order to get an impression of the general patterns of bug resolution in Firefox, we performed a survival analysis of a variety of samples of bug-reports related to Firefox. Table 1 presents the results. We carried out regressions on six samples: A sample of bugs related to Firefox in the early phases of its development; a sample of bugs related to Firefox in a later phase of its development; and for both of these samples a sub-sample of bugs started their life as New and another sub-sample for those bugs that started their life as Unconfirmed.

Bug patching is positively affected — that is, its length is statistically reduced — by direct activity in the code base, a probable proxy of the number of tentative solutions provided (Number of Commits). During the initial stage of development, it is also affected by the use of Severity and Priority levels. On the contrary, it is negatively affected by assignment and diagnosis issues, i.e. by Number Of Times Bug Was Assigned, even controlling by Log Time To First Assignment, which could also be related to the fact that Priority Was Increased, Priority Was Decreased, Severity Was Increased, Severity Was Decreased all impact negatively the patching time when they are significant — only at the later stage of development for the latter two.

Indeed, bug treatment in the early phases of Firefox appears different from treatment at a later stage. This is the case for Priority, which does not appear as an appropriate tool at the later stage, whereas the commitment of the bug reporter (Number of Edits By Bug Reporter) on the contrary appears as significant — although this last situation might also be related to self-assignment. Such as for Severity Was Increased and Severity Was Decreased, various other differences are visible in Table 1.

The difference between the treatment of bugs that were reported by outsiders — more peripheral members without the CanConfirm privilege — in contrast to the treatment of bugs in general is apparent at both stages development. “Big” bugs (Number Of Lines Of Code Added) tend to slow down the patching process when they are associated with Unconfirmed bug reports, as compared to New ones, at the Phoenix stage. On the contrary, the processing of New bugs that attract attention (Number Of Persons Copied) at the Firefox stage is slowed down, specially compared to New ones for which attracting attention appears particularly significant.

Further differences between New and Unconfirmed bugs appear at the later stage of development. First, Severity levels do not seem significant for Unconfirmed bugs.

Table 1: Significance and impact of variables controlling for bug fixing regimes (Survival analysis; Weibull fit; *** < 0.001 < ** < 0.01 < * < 0.05 < • < 0.1).

Parameter Description	Sample:	Phoenix (< 0.5)			Firefox (> 1.5)		
	All	New	Unconf'ed	All	New	Unconf.	
# Bugs	7596	6200	1366	4721	3465	1256	
Intercept	+***	+*	+**	+***	+*	+**	
Number Of Different Committers	+***	+***	+***	+***	+***		
Number Of Files Touched in Codebase	+***	+***	+***	+***	+**	+•	
Number Of Different Comments in Commits				+***	+***		
Number Of Commits	-*		-***	-***	-*		
Number Of Lines Of Code Added	-***	-***	+***				
Number Of Lines Of Code Deleted			+*				
Number Of Persons Copied	-**	-**			+*	-***	
Number Of Attachments	+*		+*				
Number Of Patches				+•			
Depends On How Many Other Bugs	+**	+*	+*			+•	
Blocks How Many Other Bugs	+*	+**					
Number Of Comments			•			+**	
Number Of Authors Of Comments	+***	+***	+***	+***	+***	+***	
Severity Trivial	+***	+***	+***	+***	+***		
Severity Minor	+***	+***	+***	+***	+***		
Severity Normal	+***	+***	+***	+***	+***		
Severity Major	+***	+***	+***		+*		
Severity Critical	+***	+***	+***		+**	-*	
No Priority	-*	-*					
Priority 1	-**	-*	-*				
Priority 2	-*					+*	
Priority 3	-*	-*				+•	
Priority 4							
Log Time To First Assignment	+***	+***	+***	+***	+***	+***	
Number Of Times Bug Was Assigned	+***	+***	+***	+***	+***	+***	
Priority Was Increased	+*	•					
Priority Was Decreased	+**	+*	•	+**	+**		
Priority Was Changed More Than Once	•	•				+•	
Severity Was Increased				+**	+***		
Severity Was Decreased				+*		+*	
Severity Was Changed More Than Once				-**			
Initial Status Was New						-•	
Bug Was Reopened At Least Once	•			-**	-**	-**	
Number Of Edits By Bug Reporter			•				
Number Of Edits By Last Assignee		+***	•		+*		
Number Of Duplicates		-*	-*	-***	-***		

Furthermore, at this stage of development, the fact that a bug starts its life as New (Initial Status Was New) implies that it is statistically solved more rapidly if it starts as Unconfirmed. But this could just be a consequence of the existence of an additional step in the life of Unconfirmed bugs, from Unconfirmed to New, precisely. When a bug originates from the periphery, from users or from less known developers, without the CanConfirm privilege, the willingness of the development community is statistically not affected whatever the number of other duplicate bug reports, that is to say, if we consider the number of duplicates as a proxy for frequency, whatever the frequency of a bug. Finally, one last element that would further provide support to the existence of users vs. innovators incentive misalignment is the significance of Bug Was Reopened At Least Once for Unconfirmed bugs which, again slightly counter-intuitively, statistically tends to speed up the development process.

6 Conclusion

In this paper, we have provided preliminary quantitative evidence for the occurrence of incentive misalignment situation between core and peripheral members of the Mozilla community at later stages of development, which we interpret as indirect evidence of “users vs. developers” incentive issues. Anecdotal evidence gathered from the analysis of several bug reports supports this view. This is all the more significant, we believe, since the Firefox project was initially thought of as specifically addressing those issues, and being intended to “Mom and Dad.”

It should be also noted, however, that developer communities are actually aware of the eventuality of such problems. For they have actually implemented voting systems in Bugzilla and in other bug repositories, from 1999 onwards in the case of Bugzilla, precisely to allow “the community”, including users and peripheral members, to vote for bugs that they would consider of particular importance. In that sense, we would believe that economists would be neither surprised of the existence of incentive issues within open-source communities, nor of the nature of the solutions and of the mechanisms implemented to contribute to their solution.

The focus of our current and future work is to strengthen the econometric side of our analysis through the exploration of alternative probability density functions to fit the data and alternative coding of some variables in the data themselves.

References

- Arora, A., J. P. Caulkins, and R. Telang (2006). Sell first, fix later: Impact of patching on software quality. *Management Science* 52(3), 465–471.
- Athey, S. and A. Schmutzler (1995). Product and process flexibility in an innovative environment. *The RAND Journal of Economics* 26, 557–574.
- Ayari, K., P. Meshkinfam, G. Antoniol, and M. D. Penta (2007). Threats on building models from CVS and Bugzilla repositories: the Mozilla case study. In *Proceedings*

- of the 2007 conference of the center for advanced studies on Collaborative research, Richmond Hill, Ontario, Canada, pp. 215–228. ACM.
- Baldwin, C., C. Hienert, and E. von Hippel (2006, November). How user innovations become commercial products: A theoretical investigation and case study. *Research Policy* 35(9), 1291–1313.
- Banker, R. D. and S. A. Slaughter (1997). A field study of scale economies in software maintenance. *Management Science* 43(12), 1709–1725.
- Bessen, J. (2002, July). Open source software: Free provision of complex public goods. *Research on Innovation*.
- Dalle, J.-M. and P. David (2007). Simulating code growth in libre (open-source) mode. In N. Curien and E. Brousseau (Eds.), *The Economics of the Internet*. Cambridge, UK: Cambridge University Press.
- den Besten, M., J.-M. Dalle, and F. Galia (2008). The allocation of collaborative efforts in open-source software. *Information Economics and Policy*.
- Estublier, J., D. Leblang, A. van der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber (2005). Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol* 14, 383–430.
- Franke, N. and E. von Hippel (2003, July). Satisfying heterogeneous user needs via innovation toolkits: the case of apache security software. *Research Policy* 32, 1199–1215.
- Garicano, L. (2000, October). Hierarchies and the organization of knowledge in production. *The Journal of Political Economy* 108, 874–904.
- Garicano, L., I. Palacios-Huerta, and C. Prendergast (2005, May). Favoritism under social pressure. *Review of Economics and Statistics* 87(2), 208–216.
- Halloran, T. J. and W. L. Scherlis (2002). High quality and open source software practices. In *2nd Workshop on Open Source Software Engineering*. ICSE.
- Kuan, J. (2005, July). Boundaries of open source software. Unpublished report.
- Lakhani, K. (2005). *Distributed Coordination Practices in Free and Open Source Communities*. Ph. D. thesis, MIT.
- Lerner, J. and J. Tirole (2002). Some simple economics of open source. *Journal of Industrial Economics* 50, 197–234.
- Lerner, J. and J. Tirole (2005, April). The scope of open source licensing. *J Law Econ Organ* 21, 20–56.
- Livingston, J. (2007). Blake Ross; creator, Firefox. In *Founders at Work: Stories of Startups' Early Days*. Apress.
- MacCormack, A., J. Rusnak, and C. Y. Baldwin (2006, July). Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science* 52(7), 1015–1030.

- McHugh, J. (2005, February). The Firefox explosion. *Wired* 13(2).
- Mendonca, L. T. and R. Sutton (2008, January). Succeeding at open source innovation: An interview with Mozilla's Mitchell Baker. *The McKinsey Quarterly*.
- Raymond, E. (1999). The revenge of the hackers. In *Open Sources: Voices from the Open Source Revolution*, pp. 207–219. Sebastopol, CA: O'Reilly.
- Ryan, O. (2006, June). Interview with Blake Ross. Archived at <http://mozillamemory.org>.
- Sack, W., F. Détienne, N. Duchenaud, J.-M. Burkhardt, D. Mahedran, and F. Barcellini (2006). A methodological framework for socio-cognitive analyses of collaborative design of open source software. *Computer Supported Cooperative Work* 15(2-3), 229–250.
- Villa, L. (2003). Large free software projects and bugzilla: Lessons from GNOME project QA. In *Proceedings of the Linux Symposium*, Ottawa, Canada.
- von Hippel, E. (2005). *Democratizing Innovation*. MIT Press.