Different Bug Fixing Regimes? A Preliminary Case for Superbugs *

Jean-Michel Dalle[1] and Matthijs den Besten[2]

[1] Université Pierre et Marie Curie, Paris, France; jean-michel.dalle@upmc.fr
[2] University of Oxford, Oxford, UK; matthijs.denbesten@oerc.ox.ac.uk

Abstract. The paper investigates the processes by which bugs are fixed in open-source software projects. Focusing on Mozilla and combining data from both its bug tracker (Bugzilla) and from its CVS, we suggest that: a) Some bugs resist beyond the first patch applied to the main branch of the source code in relation to them, which we denote as superbugs; b) There might exist different bug fixing regimes; c) priority and severity flags as defined in bug repositories are not optimized for superbugs and might lead to a involuntary side effects; d) The survival time of superbugs is influenced by the nature of the discussions within Bugzilla, by bug dependencies and by the provision of contextual elements.

There have always been claims according to which open-source software would be structurally able to implement more efficient development methodologies along several dimensions, and notably vis-à-vis reliability. "Given many eyeballs, all bugs are shallow" [4]. In other words, the visibility and accessibility of the source code, in addition to extended peer review processes associated with the existence of a community, would be key to a superior reliability.

However, it is probably honest to say that these claims have never received a true empirical validation, although most data that relate to open-source development is archived online and thus freely available to both open-source projects and academia. Furthermore, a recent study by Coverity[2], a Stanford spin-off company whose technologies allow for the automatic analysis of source code to identify some of its defects, has stressed that mission- and safety-critical proprietary applications were able to reach reliability levels above the most reliable open-source projects. Beyond the straightforwardness of this finding, it still contributes to upgrade the general view on the reliability levels that different open-source projects were able to reach, while quality and reliability are becoming a major concern for open-source software.

Are there indeed general elements in the open-source development methodology that would allow for an intrinsic superiority of open-source in terms of

* The work presented here has benefited from discussions with Patrick Brézillon, Paul A. David, Laurent Daudet, Fabrice Galia, Hela Masmoudi and others. The support of Calibre, an EU FP6 project, and of NSF project NOSTRA in the early phases of this research are also gratefully acknowledged.

[2] http://scan.coverity.com/

Please use the following format when citing this chapter:

Dalle, J.-M. and den Besten, M., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 247–252.

reliability and defect density? There is a strong case here for empirical studies, in a more general context of inquiries about open-source software development based on mining available online archives. Indeed, bug fixing processes in open-source software have already received some attention [1, 2, 3, 5]. Following this literature, we further suggest that more could be learnt by mining data from *both* bug trackers *and* code repositories, i.e. the interactions between what happens in Bugzilla and what happens in the CVS. We suggest that there might exists several different fixing *regimes*, and that, among the bugs that tend to resists beyond the first attempts to fix them, there exist superbugs — as we propose to denote them, in a direct analogy with antibiotic-resistant bacterias — for the fixing of which the exchange of context elements between users and developers might be key.

Data We focus here on Mozilla, not only as a prominent example of a successful open-source software project that has already been the subject of various empirical research investigations, but also because it is in the context of Mozilla that the well-know bug repository Bugzilla originates. We combine Mozilla CVS data with Bugzilla data. Namely, for each bug number in Bugzilla, we look for this number in commits to the main branch of Mozilla's CVS, using a heuristic script that we have developed to this purpose. We then combine, for each bug number, some of its characteristics inferred from Bugzilla with other characteristics inferred from retrieved CVS data.

We limit ourselves to relatively old data to avoid censoring biases: i.e. we suppose that a sufficiently long time has elapsed so that we can neglect bugs that would not have been fixed yet. We also did some preliminary cleaning up of the database, removing the first 1000 smaller bug numbers, so as to avoid transitory initial conditions and to control the fact that smaller number have a higher probability to be found in the CVS using our script while not corresponding to bug numbers. As a result of preliminary univariate analysis, we removed some outliers, and typically removed from our sample all bugs that would correspond to either more than 1000 files fixed, that would depend from more than 25 other bugs, and that would be associated with a bug report open in Bugzilla for more than 1500 days before the first patch associated with the corresponding bug number is committed to the CVS. Furthermore, we removed from most of the analysis presented in this paper all bugs whose severity had been set to "enhancement" as they would rather correspond to feature requests properly speaking. Finally, we limited ourselves to all bugs whose resolution had been set as fixed, compared to others resolution types. Ultimately, our database includes approximately 17000 bugs.

Superbugs First of all, many bugs tend to "live" for a long time after the code is first patched in relation to them. There are for instance 650 bugs in our sample that were patched between 10 and 100 days after the first patch was applied in relation to them to the main branch of the code base. Compared to previous studies, these bugs are not simply associated with long discussions in bug tracking system, but also with patches associated with them appearing for

a long period of time in the code main branch. Although a considerable fraction of the bugs are corrected on the first day in which the code of the main branch is patched in reference to them, this "long tail" effect is however important. That is to say, some bugs seem to be resistant to the "treatments" that they initially receive: in that sense, we suggest to call them *superbugs*, in an analogy with other resistant life forms that are now developing in hospitals. Compared to other unusual software bugs such as Heisenbugs, superbugs belong to a more general category that might include some of these more peculiar ones.

Moreover, there seem to exist different *fixing regimes*: The hazard function that fits to our data (details upon request) has a 'bathtub' shape. A shape well-know in engineering and generally associated with the existence of 3 different regimes: an initial phase, a flat middle one, and a last so-called 'wear out' one — in our case below 10 days, between 10 and 100 days, and above 100 days. Within the latter category, it might be that there would even exist another regime above 1000 days. It is absolutely clear to us that this categorization is very tentative, being based only on one open-source project, and we very much hope that future investigations will refine it. However, as a first step to progress in the exploration of bug fixing regimes, we suggest to denote bugs fixed in 10 days or less after a first patch has been applied in the main branch in relation to them simply as *resistant* bugs, while bugs fixed in more than 10 days could be characterized as *superbugs*. And since the latter category could itself include different regimes, we will denote as *hyberbugs* bugs fixed in more than 100 days and 1000 days or less, leaving bugs fixed above 1000 days for future investigations.

Fixing Time: severity and priority flags What factors affect the fixing time of bugs in all three regimes? Obvious candidates are the two variables that are set by bug reporters and developers, respectively, to characterize bugs, namely, severity and priority. severity is set by bug reporters under explicit guidance not to use the blocker and critical levels (severity = 6 and 5, respectively) out of purpose, while the variable priority is set by developers. Our analysis of our data (available upon request) shows that there is no distinguishable pattern, except for severity = 1 (trivial) and 2 (trivial). trivial would seem to result globally in a relatively lower survival probability while minor results in a relatively higher one. These results are confirmed when plotting similar graphs for each of the resistant bug, superbug, and hyperbug regimes: however, more precisely, the minor effect is more apparent for superbugs and hyperbugs, and the trivial effect for resistant bugs. One hypothesis here is that, due also a limited number of bugs associated with severity 1 and 2 that tends to show that these two categories aren't used very often by developers, trivial bugs might be eliminated more rapidly precisely because of their triviality i.e. *easiness to correct*, thus in the resistant bug regime, whereas bugs flagged as minor would on the contrary could tend to be *neglected* compared to others of higher severity, an effect that would naturally be more pronounced as times goes on, i.e. for superbugs and hyperbugs. If so, it might be worth simplifying the number of severity categories in bugzilla

by typically avoiding minor and flagging bugs as trivial or normal or higher. Furthermore, the only distinguishable pattern for priority is for priority = 5 (P5): bugs flagged at a very low priority tend to be patched earlier. This surprising finding is probably to be related to a different use of priority flags under different regimes. There are no P5 bugs among hyperbugs. An explanation for this would be that working on bugs of low priority is abandoned: not that these bugs are necessarily fixed, but they do would not survive for *lack of interest*. The fact that the priority variable is set by developers, compared to the severity variable by bug reporters, would tend to support this explanation.

Fixing Time: survival analysis It is possible to fit predictive models of the fixing time of bugs in all 3 regimes. The linearity of the survival function in loglog vs. log scale suggest that using a Weibull distribution approximation is reasonable as a first step, although the actual distribution is most probably of a power-law or similar nature at least for resistant bugs and superbugs Table 1 synthesizes results of survival analysis regressions (detailed results available on request).

First, both priority and severity appear effective for resistant bugs, and less so for superbugs and hyperbugs. Some of the priority levels influence survival time counter-intuitively on the full sample, which we again interpret as resistant bugs and superbugs with low priorities being abandoned. The highest two levels of priority (and when priority is missing) tend to reduce the survival time of superbugs compared to P5. With due cautiousness due to the improper fit of the model, two levels of severity, minor and major, seem to have an influence in increasing and decreasing, respectively, the survival time of hyperbugs compared to blocker level. The minor effect, already presented above, is indeed also present on the full sample. The interpretation of the negative major effect on hyperbugs and of the negative critical effect on the entire sample are less clear. More interestingly perhaps, different levels of severity influence the survival time of bugs differently: normal has a less pronounced effect than critical, major and minor, compared again to blocker. Two different effects might be at play here: a minor effect, again, associated with neglect, and maybe a difficulty effect, critical and major being just more difficult to fix or implying more cautiousness, discussions and care.

Second, both resistant bugs and superbugs are affected by dependencies, again counter-intuitively: bugs that block many other bugs appear to be fixed *less* rapidly, while bugs that depend upon many others are fixed *more* rapidly. We interpret the first part of this finding as a probable consequence of the difficulty of fixing bugs that block many others: that is, the fact that a bug blocks several others indirectly is an evidence of interdependencies that render its fixing lengthier. The second part of this finding is less clear. It might be, since *bug report networks* play an important role in bug fixing processes as has been recently suggested [5], that bugs inserted in bug report networks would tend to attract more *attention* from developers: and dependent bugs would be fixed relatively rapidly once the bugs that blocked them would have been fixed,

Table 1. Significance and impact of variables controlling for bug fixing regimes. The source of the data is indicated with B for Bugzilla and C for CVS.

Parameter	Bugs	SuperBugs	HyperBugs	Full Sample	(Parameter Description)
#Bugs	25965	650	290	16924	
Intercept	(−)***	(+)***	(+)***	(−)***	
tpsStart^{BC}	(−)**			(+)***	(first commit − opening date)
nauth^{C}	(+)***	(+)***	(+)***	(+)***	(maintainers mentioning bug)
nfile^{C}	(−)***			(−)***	(files touched by bug-commits)
ncome^{C}	(+)***			(+)***	(commit-comments per bug)
ncomi^{C}	(+)***			(+)***	(commits per bug)
sadd^{C}	(−)**	(−)**		(−)**	(added lines of code per bug)
sdel^{C}		(−)**		(+)**	(removed code per bug)
ccsz^{B}	(−)***			(−)***	(# addresses in cc-list)
attac^{B}		(−)**			(attachments per bug)
patc^{B}		(+)***			(attachments marked "patch")
depen^{B}	(−)***	(−)**		(−)***	(bug dependencies)
bloc^{B}	(+)***	(+)**		(+)***	(bug blocks)
comm^{B}					(number of comments)
comau^{B}	(+)*			(+)***	(# commentators)
comli^{B}					(# lines of comments)
priority^{B} 0	(−)***	(−)*		(+)***	
priority^{B} 1	(−)***	(−)*		(+)***	
priority^{B} 2	(−)***	(−)**		(+)**	
priority^{B} 3	(−)***			(+)***	
priority^{B} 4	(−)***		•		
priority^{B} 5	•	•	n/a	•	
severity^{B} 1					
severity^{B} 2	(+)***		(+)**	(+)*	
severity^{B} 3	(+)**				
severity^{B} 4	(+)***		(−)*		
severity^{B} 5	(+)***			(−)**	
severity^{B} 6	•	•	•	•	

at least compared to all bugs that do not depend upon any other and specially bugs that are not part of a bug network.

Third, the number of developers (nauth) contributing to the code is positively related to survival time. Resistant bugs to which more numerous and different commits are related (ncome, ncomi) also tend to be fixed less rapidly, and similar effects hold for resistant bugs when more numerous people participated in the bugzilla discussion (comau). On the contrary, tpsStart, i.e. the length of the discussions and experimentations before a first patch is applied to the main branch, reduces the survival time of resistant bugs. Similarly, ccsz (number of developers who were copied when updates were made to the bugzilla system) also reduces the survival time of resistant bugs. Both of these effects disappear for superbugs. On the full sample, ccsz is also significant, and still

negative, while tpsStart is significant, but positive. Evidence is therefore mixed here about the effect of "eyeballs" on fixing time. The number of active participants in the discussion tends to slow down fixing, but this might just reflect how complex to fix a bug is. Conversely, longer discussions (tpsStart) and the number of observers (ccsz) would tend to allow for a solution to be found more rapidly at least for resistant bugs.

Fourth, the fixing of superbugs is not affected by most of the variables that influence fixing resistant bugs. On the contrary, they are sensible to the number of lines deleted — maybe as a consequence of simply removing the part of the code that created a superbug? — and also to the number of attachments and to the number of patches sent in the Bugzilla discussion: the higher the number of patches sent, the longer it takes to fix a superbug; conversely, the higher the number of attachments, the lower the survival time of a superbug. A straightforward interpretation for the former finding is that the number of patches might be a consequence of how difficult it is to fix a given superbug. About the puzzling latter finding, it should be noted that attachments are often screen captures and other contextual elements: a superbug would then tend to be fixed more rapidly as soon more elements of context would be contributed to the discussion. This would be for instance in line with the fact that intermittent failures generally tend to be difficult engineering problems, whose intermittence is often due to missing *contextual* elements. An open question here is whether the provision of contextual elements earlier during the bugzilla discussion would have prevented the transformation of normal bugs into superbugs.

Conclusion We believe that understanding how superbugs could be fixed more rapidly could be of special relevance vis-à-vis the reliability of open-source software. priority and severity variables as currently defined in bug repositories do not appear optimized yet in this respect. We suggest that analyzing the formation of bug report networks, clarifying the nature of the discussion in bug repositories between participants assuming different roles, and understanding how contextual elements are brought into bug repository discussions are also interesting research avenues.

References

1. K. Crowston, J. Howison, and H. Annabi. Information systems success in free and open source software development. *Software Process*, In Press.
2. K. Crowston and B. Scozzi. Coordination practices for bug fixing within FLOSS development teams. In *Proc. CSAC*, 2004.
3. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development. *ACM Trans. Softw. Eng. Methodol.*, 11:309–346, 2002.
4. E. S. Raymond. The cathedral and the bazaar. *First Monday*, 3, 1998.
5. R. J. Sandusky, L. Gasser, and G. Ripoche. Bug report networks. In *Proc. ICSE Workshop Mining Software Repositories*, 2004.