2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement

Impact of Triage: a Study of Mozilla and Gnome

Jialiang Xie*, Minghui Zhou* and Audris Mockus†
*School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
{xiejl11@sei.,zhmh@}pku.edu.cn
† Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

***Abstract*—Triage is of great interest in software projects because it has the potential to reduce developer effort by involving a broader base of non-developer contributors to filter and augment reported issues. Using issue tracking data and interviews with experienced contributors we investigate ways to quantify the impact of triagers on reducing the number of issues developers need to resolve in two OSS projects: Mozilla and Gnome. We find the primary impact of triagers to involve issue filtering, filling missing information, and determining the relevant product. While triagers were good at filtering invalid issues and as accurate as developers in filling in missing issue attributes, they had more difficulty accurately pinpointing the relevant product. We expect that this work will highlight the importance of issue triage in software projects and will help design further studies on understanding and improving triage practices.**

I. Introduction

Developing source code is not enough for a successful software project. Necessary levels of quality and completeness of functionality could be realized only through feedback from a large population of contributors, especially users. Such input is typically managed via issue tracking systems (ITS). A small core team [1] of popular OSS projects can be easily overwhelmed by the massive inflow of issues ($>$ 50K/year in Mozilla and Gnome) leading to delays, unsatisfied users, and lower quality of the product. In particular, a considerable number of reported issues are of low quality and can not be fixed. For example, issues lacking sufficient information represent 15% and 6% of all resolved issues in Gnome and Mozilla [2]. It is, thus, paramount to recruit, train, and retain a broader base of contributors (triagers) to filter and augment this predominantly low-quality input, so that the developers can spend their time on fixing real issues. Triage as a procedure to filter and improve the quality (relevance, accuracy, reproducibility, non-duplication, and completeness) of user reported issues is, thus, a critical part of any large project.

Many contributors help with issue reporting and resolution. For example, Gnome project Evolution had more than 20K contributors helping to report and resolve issues, but only 1K developers contributing code over the past decade. However, the value of contributions made by these non-developer triagers may be underestimated. A developer who does a triage on an issue, can also fix the issue, unlike the non-developer triagers. Consequently, non-developer triagers may feel unappreciated, e.g., a triager left Mozilla "because of a general lack of interest in doing anything substantial to improve the triage process"[3].

For brevity, we use the word "triager" to refer to a non-developer triager in the remainder of this work.

We aim, therefore, to help such communities by understanding the value triagers may bring and how to leverage their strength to improve the project. In particular, we investigate:

- What are triage activities in Mozilla and Gnome? Do they differ?
- With what activities do triagers help the most?

We introduce the methodology in Section II, and present the results in Section III. We discuss the validation and limitations in Section IV, describe the related work in Section V, and conclude the paper in Section VI. We provide the data and scripts we used at *http://www.passion-lab.org/projects/triage.html*.

II. Methodology

We study more than a decade of data for Gnome and Mozilla in the corresponding ITS, i.e., Bugzilla systems in this case. We follow procedures to analyze such data described in [4], in particular, we iterate over the following steps to increase the quality of data: first we retrieve the raw data, then perform initial cleaning and processing, create measures to answer our research questions, perform analysis of these measures, and finally validate the results. The validation step involving experts from both projects led to revisiting and modifying assumptions made in the earlier steps and resulted in several iteration.

A. Bug Triage Protocol

We found many documents describing bug resolution process. Gnome provides its triage protocols [5], and Mozilla has a triage guide [6]. In a nutshell, an issue (often referred to as bug) that needs a triage starts from UNCONFIRMED status. A contributor, for example, a triager or developer, picks such "open" issue and then either 1) indicates that the issue is valid and needs developer's attention by setting the status to NEW, or 2) closes the issue by setting the status to RESOLVED with one of the resolutions shown in Table I.

To understand the triage protocol better, we had several email exchanges with four contributors (a bug-master from

*Corresponding author

978-0-7695-5056-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ESEM.2013.62

TABLE I: Misconfirmed Reports with Their Final Resolution

Project	Duplicate	WorksForMe	Invalid	WontFix	Incomplete	Exprd/Obslt	NotABug	#Issues
Mozilla	9452/39.7%	7981/33.5%	2355/9.9%	2328/9.8%	1045/4.4%	637/2.7%	N/A	23800
Gnome	2924/33.0%	N/A	1389/15.7%	820/9.3%	1941/21.9%	651/7.4%	535/6.0%	8861

Gnome and an experienced developer from Mozilla and two ordinary triagers, whom we refer to as gnome-1 and 2, and mozilla-1 and 2), clarifying our interpretation of triage activities and the issue tracking data. As our understanding of triage activities and impact increased, we followed up with a more focused analysis and more specific questions.

B. Preparing Data

We first retrieved Bugzilla data from Gnome and Mozilla in March of 2011. For both communities, we removed the issues prior to and including 2000 because of the data quality of the time-stamps for issues reported during that early period.

As noted in II-A, we define as triage activities confined to issue workflow between status UNCONFIRMED and RESOLVED/NEW. We consider two types of activities as triage: modifications to issue attributes and, issue confirmation (change of status to NEW or RESOLVED). Data associated with each triage activity includes issue ID, the date of activity, the login of the actor, the name of the modified attribute or "status" for confirmation, its old value, and the new value.

Note that some issues may not have any triage activities, because they're submitted by the contributors with the privilege to set the initial status to NEW (e.g., developers).

The dataset we analyzed contains 1,153k triage activities in 397k issues of Gnome and in 1,492k triage activities in 249k issues of Mozilla.

C. Identifying the Roles

Triagers are not the only contributors who conduct triage activities, as other contributors, e.g., developers also conduct triage. To quantify the impact of triagers and to compare the accuracy of their activities to the accuracy of developers, we need to determine who is a triager and who is not. Based on the analysis of contribution profiles in Bugzilla and Version Control System (VCS) we were able to identify two additional roles involved in triage activities: developers and issue reporters.

In particular, we identify *Developers* in Bugzilla by matching them to code committers in each project. We operationalize *Triagers* as contributors who conduct at least one triage activity on issues that were reported by another person. A contributor may change their role over time, we, therefore, consider a person to be a triager only during the period before their first code commit. The contributors who modify only issues that they have previously reported we assigned to *Reporter* role.

D. Accuracy of Triage

In addition to the number of activities triagers perform and the number of issues they participate in, we also would like to know if they are accurately performing these tasks. We measure the accuracy of a triage activity by determining if the attribute value it sets for an issue is the "correct" value. Because there is no "gold standard" for the correct value of an attribute in an issue tracking system, we focus on counting likely mistakes. We can not determine mistakes in triage activities with no subsequent action on the same issue: without another person inspecting the issue it is not possible to tell if a mistake was made. The remaining triage activities we consider to be a "mistake" if they set an issue attribute to a value that is different from the final value of that attribute. In particular, we count mistakes only for issues that were resolved and satisfy at least one of the conditions: the issue was fixed; the issue was confirmed; or there were subsequent activities on the same issue.

III. Results

We outline our findings on triage activities in Mozilla and Gnome and discuss the differences between the two projects in Section III-A, and quantify the impact of triage activities in Section III-B.

A. What are triage activities in Mozilla and Gnome

Generally, the nature of bug triage is to harness the incoming bug reports. There are three kinds of triage tasks.

First, check the relevance of the report. As Mozilla-1 described, "triage is basically the process of filtering incoming bug reports", it aims to answer, "is this bug report actually a bug, or is it something else, spam, a third-party program, support request, etc.?" This task, therefore, is to confirm relevant reports and to reject irrelevant reports. We consider resolving non-reproducible, or not-relevant issues as one of the benefits triagers provide and refer to it as *filtering*.

Second, complete the report information, in particular, complete the attributes like Severity, Priority, Product, OS, and Version. That is, as Gnome-1 emphasized: "Triagers make sure that reports include enough information to be useful for developers." They greatly help developers because, as Mozilla-1 pointed out: "Getting complete information takes the most time as it often requires a back-and-forth of communication between the triager and the reporter."

Third, determine the location of the report: "Is this bug in the right product so it will be seen by the right developers?" – as proposed by Mozilla-1. Gnome-1 also explained this task: "One of the triagers' task is to assign reports to products, not specified logins." The contributors who do the triage "normally never change the assignee manually", he commented, "When we set up a new product in GNOME Bugzilla we create a 'virtual' default assignee in the form of 'productname-maint@gnome.bugs'. We ask developers of the product to add this account to their 'User Watchlist'. If a developer plans to work on a bug report, she can assign the bug report to herself."

In summary, triager's three tasks are to filter (confirm or reject), fill information (complete attributes such as Severity, Priority, OS and Version), and assign products for reported issues.

TABLE II: The Number and Proportion of Issues with a Modified Attribute

Project	Severity	Priority	Version	OS	Product	Total Triaged Issues
Gnome	18K/4.4%	13K/3.3%	25K/6.2%	7K/1.7%	13K/3.2%	397K
Mozilla	18K/7.4%	4K/1.5%	40K/16.2%	19K/7.5%	22K/8.9%	249K

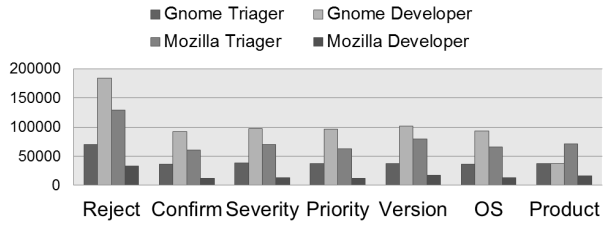

Fig. 1: Number of Issues by Triage Task and Role

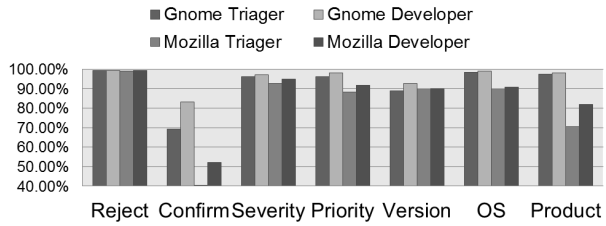

Fig. 2: Accuracy of Issues by Triage Task and Role

Table II shows the number and proportion of modifications to various attributes of triaged issues in both communities. In particular, we see that triagers modify Product, OS, Version and Severity in a larger fraction of triaged issues for Mozilla and Priority in a larger fraction of triaged issues for Gnome.

From the reviews and on-line documents and by inspecting a sample of relevant issues, we found two reasons for these differences. First, the user base is different. Mozilla has a much larger base of users, thus an average user is likely to have less computer expertise than an average Gnome user, e.g., "Many bugs get marked Firefox when they are really bugs in the core engine." Such broad base of users also lowers the quality of reports and requires triagers to add sufficient information to make them useful for developers.

The second reason we found stems from the differences in community policy. For example, Mozilla Triage Guideline notes: "don't change Priority field, which is for the developer" [6]. This may partly explain why Priority field is changed less in Mozilla.

B. Impact of Triagers in Mozilla and Gnome

We evaluate triager's contribution by the number of modified issues and the accuracy of the information they provide to developers. Given that a triager may change several attributes of an issue, or change the same attribute of an issue several times, we calculate triager's contribution by the number of modified issues. To measure accuracy, we consider the fraction of activities without "mistakes". We also compare triagers' contribution to that of developers'.

Figure 1 illustrates the number of issues for different types of triage tasks and Figure 2 shows the quality of different types of triage tasks conducted by triagers and developers in both communities. Note Severity, Priority, Product, OS, and Version represent the task of completing the corresponding attribute.

Figures show that the biggest impact of triagers was on filtering reports. In particular, the number of issues filtered by triagers was $190K$ (77% of all filtered issues) in Mozilla, and $106K$ (27% of all filtered issues) in Gnome. Meanwhile, triagers rejected issues accurately: in both project, 99% of the rejections were correct, higher than for any other type of task.

Triagers had the second largest impact on completing information for newly reported issues, in particular, they completed attributes for OS, Version, and Severity for a large number of issues and accurately. Gnome-1 commented on this finding: "this information is meta data and can be easily asked for and corrected in one step." It suggests that completing the basic information may be a good starting point for beginner triagers.

Figure 2 shows that the triager product assignments are often incorrect. It seems to be particularly difficult in Mozilla: "It can be an issue in the underlying stack instead of the application, and finding the exact low-level library is hard for an average triager."

Table I shows that a large fraction of triager-confirmed issues are not fixed with 59% and 30% of issues confirmed by triagers were not fixed (incorrectly confirmed) in Mozilla and Gnome respectively.

The largest fraction of such issues comes from duplicate reports. The projects use some duplicate-detection technique, e.g., Mozilla Crash Reports [7] and Gnome Duplicate-finder, however, these techniques "are not perfect" [8], especially for those without stacktraces such as UI, enhancements or translation problems. Therefore, triager has to search through existing reports, including the issues that are already fixed or identified as duplicates.

Therefore, we expect that triagers may be most effective in reducing developers' load by filtering irrelevant reports, and by filling in missing information. But confirming issues and assigning them to the correct products are not easy tasks for triagers.

IV. Validation and Limitations

As mentioned earlier, we follow procedures described in [4] to ensure that we accurately interpret Bugzilla data and measures derived from it.

In particular, we had to iterate to arrive at the method we used to separate triagers from the other roles based on their activity profiles. While it is relatively easy to identify developers via their commits in the VCS, contributors may change their role over time, with many triagers eventually contributing code (thus becoming a developer). We, therefore, assigned a sequence "(role, time)" of tuples to each individual based on the time when the role was first assumed in the dataset. And we only considered a contributor as a triager before she became a developer. Meanwhile, Reporters may conduct

"triage" activities because anyone is permitted to modify the issues they report. We, therefore, considered contributors to be triagers only if they triaged issues reported by others.

A particularly vexing problem we faced was the measure of accuracy for a triage activity. After many iterations and experimentation with alternative measures we arrived at the presented heuristic even though it has a number of drawbacks. For example, typically the severity of a NOTABUG issue is not considered to be important, thus its final value may not be verified. We, therefore, did not evaluate accuracy of activities with no subsequent activities for the same issue, because the attributes modified by these activities may not have had a chance to be looked at (and, thus, may not have been verified).

We observed that some products changed their name. We, therefore, do take it into account in calculating product assignment accuracy. For example, if the final product value is B, but the triager assigned the issue to product A. We do not consider it to be a mistake if the product A was renamed to B. To discover all instances of product renaming we searched for inconsistencies in product names data and validated these as instances of renaming by confirming via documents describing the history of that product.

V. Related Work

There has been a substantial amount of work on trying to understand and help improve the issue/bug tracking practices.

"Who can fix this bug?" is an important question in bug triage needed to "accurately" assign developers to bug reports. For example, machine learning techniques [9], [10], and graph model based on Markov chains [11], were used to better assign developers to bug reports.

Detecting duplicate (or similar) reports is another common topic, with studies attempting various ways to compare the similarity of issue reports. For example, Natural Language Processing (NLP) techniques [12], and NLP techniques plus execution information [13] were used to suggest the most similar bug reports to the new bug report.

Bug fixing is an important topic with a substantial amount of literature. Common questions include: which bugs get fixed [14]? how long it takes to fix a bug [15]?

Unlike prior work, this study focuses on understanding and quantifying triagers' practice, in particular, what tasks a triager is involved with and what value triagers may bring to the project.

VI. Conclusion

We study the nature of triage and the impact of non-developer triagers by analyzing the issue tracking practices in Gnome and Mozilla.

We found that the critical triage goals are to filter incoming reports, fill incomplete information, and assign reports to products. We also found some variations in triage practice between the project that may be attributable to different policies and user base. Our analysis shows that non-developer triagers make a substantial contributions by filtering non-fixable issues and by filling basic information, while confirming issues and assigning products appear to be a difficult challenge for triagers.

We plan to use these results to investigate the reasons for the difficulties triagers face in product assignement and in confirming issues and we plan to develop tools to support such tasks. In particular, we plan to produce a tool that would help triage practice by predicting the accuracy of all attributes in an issue report and would suggest ways to correct it. We hope that our work would be worthy of the following quote from triager's guide: "By following these techniques you will help the important bugs get fixed, as well as optimize precious developer time."

VII. Acknowledgment

Partial support by the National Natural Science Foundation of China Grants 91118004, 61121063 and 61073016.

References

[1] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 1–38, July 2002.

[2] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in open source community," in *ICSE 2012*, Zurich, Switzerland, June 1–9 2012, pp. 518–528.

[3] T. Downer, "Some clarification and musings," http://tylerdowner.wordpress.com/2011/08/27/some-clarification-and-musings, 2011.

[4] A. Mockus, "Software support tools and experimental work," in *Empirical Software Engineering Issues: Critical Assessments and Future Directions*, V. Basili and et al, Eds. Springer, 2007, vol. LNCS 4336, pp. 91–99.

[5] "The gnome bugsquad," https://live.gnome.org/Bugsquad, 2012.

[6] "Mozilla triage guide – harnessing the flood of community," https://wiki.mozilla.org/QA/Triage, 2010.

[7] "Mozilla crash reports," https://crash-stats.mozilla.com.

[8] "Finding duplicates," https://live.gnome.org/Bugsquad/TriageGuide/FindingDuplicates, 2010.

[9] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 361–370.

[10] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 10:1–10:35, Aug. 2011.

[11] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. New York, NY, USA: ACM, 2009, pp. 111–120.

[12] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 499–510.

[13] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 461–470.

[14] P. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, vol. 1, may 2010, pp. 495 –504.

[15] S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *Proceedings of the 2006 international workshop on Mining software repositories*, ser. MSR '06. New York, NY, USA: ACM, 2006, pp. 173–174.