How much time did it take to notify a Bug? Two case studies: ElasticSearch and Nova

Gema Rodriguez-Perez
GSyC/LibreSoft
Universidad Rey Juan Carlos
Fuenlabrada, Spain
gerope@libresoft.info

Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
Fuenlabrada, Spain
grex@gsyc.urjc.es

Jesus M. Gonzalez-Barahona
GSyC/LibreSoft
Universidad Rey Juan Carlos
Fuenlabrada, Spain
jgb@gsyc.es

***Abstract*—The *Time To Notify* (TTN) a bug is a valuable metric in the software maintenance and evolution studies that describes how much time it takes for a bug to be notified/reported in the issue tracking system since the time the bug was introduced into the source code. Even so, it is still a challenge to exactly calculate it since no precise way exists to locate where and when the bug originated. This paper aims to study what is the value of TTN in two different projects. For a set of bugs in these projects, we know exactly which "previous commit" was the cause of the failure in the system. Furthermore, to better understand how this is related to the maintenance and evolution of a software, we also analyze the relationship between TTN and other metrics extracted from the source code management (SCM) system such as the author of the bug, the *Time To Fix* (TTF) or the developer experience. We have observed that the mean of the TTN in the projects was 312 days and 431 days. However, only one of the projects showed a moderate correlation between the experience of the author who created the bug and TTN.**

***Keywords*-Bug introduction change; bug seeding metrics;**

I. Introduction

It is well-known that a large amount of the total effort of a software system is spent in maintenance and evolution [20], some sources even assigning this phase over 80% of the total cost. Hence, researchers have spent much effort understanding and characterizing software maintenance and evolution processes. One of the main areas in software maintenance and evolution is related to bug detection and analysis. Metrics such as the *Time To Fix* (TTF), the *Time To Review* (TTR) or the *fix-effort* to improve code quality have been thus proposed in the research literature [11][14]. One of these metrics, the one that this paper is going to analyze is the the *Time To Notify* (TTN) a bug. The TTN is the time that goes from the time where a bug has been introduced in the source code to the time when the wrong behavior is notified in the bug tracking system of a project.

The concept of TTN provides insight into fundamental aspects of the software maintenance of a project that are not easy to obtain, especially in Free/Open Source Software (FOSS) communities – such as the well-known Mozilla, Eclipse or GNOME projects. In these environments, effort information has historically neither been stored nor maintained in their bug tracking system (e.g., Bugzilla) in spite of the valuable knowledge of the product complexity that it could provide. Currently, the only bug tracking system that allows store and maintain some type of effort information related to bug fixing activity is JIRA [21], as it includes an estimate of the time it will take to fix the issue. However, the TTN can be potentially calculated by combining information from the source code management system (SCM) and bug tracking system (BTS). Although it won't directly provide a measure of (human) effort, it is a proxy of efficiency: low values of TTN may indicate better maintained software (with code being changed/tested continuously), while higher values may denote less maintained software (with potential risks being present for longer periods of time).

To compute the life of a bug in a software, it is necessary to know exactly when and where a bug was introduced. As versioning systems (such as git) are commonly used in modern software development, meta-data (committer, author, time, etc.) on all changes exists, and we could potentially identify what line introduced the bug. However, finding the line of code where the bug was introduced is not a trivial task. Several methods have been proposed to determine what commits are the candidates of having introduced a given bug. The popular SZZ algorithm [19] is one of them: it traces the lines touched in a *fixing* commit back to the time when these lines were modified or added. The main concern in using this algorithm is the that it is prone to provide a wrong measure in some situations that frequently happen during the evolution of a software system. These situations, such as changes in the API, present a common characteristic: the bug was not introduced in the previous commit(s). So, when the line was introduced, the software was correct; the wrong behavior appears due to other, external changes.

In this paper, we measure TTN for a set of bugs from two different FOSS projects: ElasticSearch and Nova. Because of previous research performed on them by the authors, we know the exact location of the *Bug Introducing Change* (BIC) for a set of bugs. This means that from a set of "previous commits" of each of the fixing lines, we are able to identify what specific previous commit caused the failure in the system. We compare as well our measurements of TTN with the ones obtained with the SZZ algorithm.

Furthermore, to better understand how this is related to the maintenance and evolution of a software, we also analyze the

relationship between TTN and other metrics extracted from the SCM system, such as the author of the bug, the time to fix or the developer experience inserting the bug.

Summing up, we are interested in studying the *real* value of TTN to answer following research questions:

- RQ1: What are the values of TTN in both projects? What is its mean, median, etc.?
- RQ2: What is the correlation of TTN with others characteristics of the project/developers involved in the bug fixing process?

Our results reveal that the mean TTN differs from the mean calculated with other methods. In addition, we have found that TTN has a dependency with the experience of the developer who introduced the bug. On the other hand, we note that to measure the TTN is crucial to determine exactly the location of the BIC; so far, now no precise way exists. Thus, we propose a simple idea which may shed some light to this problem.

The remainder of the paper is organized as follows: In Section II the current body of knowledge is presented. Next, Section III describes the methodology used to identify the Bug Introducing Change (BIC) and calculates the metrics. Results obtained for Nova and ElasticSearch are presented in Section V. Section VI answers the research questions, and discusses potential applications and improvements to our approach. After reporting the limitations and threats to validity in Section VII, we draw some conclusions and point out potential future work in Section VIII.

II. Related Work

Kim and Whitehead calculated the time to fix bugs (*bug-fix time*) in ArgoUML and PostgreSQL projects reporting that the median time for fixing a single bug is around 200 days. Also, they argue that this time is a significant factor to measure the quality of a software system [11]. Furthermore, Guo *et al.* studied two Microsoft products, looking for the attributes that had influence in the fixing commit: they found that bugs reported by developers with higher reputation were not only more likely to get fixed, but also fixed faster [7].

Eyolfson *et al.* define the (*bug-fix time*) as the time from the earliest commit that introduced the bug to the bug-fixing commit. Their findings show that the time and day of a code commit may affect the quality of the code [4].

Lionel *et al.* also studied the fix-time for Bugs in large FOSS projects; their results indicate that the priority of a bug in Eclipse is correlated with the time to fix [12]. On the contrary, Bhattacharya *et al.* used regression testing to measure the correlation between bug-fix time with some of the bug report attributes such as number of attachments or bug severity, finding that their results did not present such correlation [3].

Zhang *et al.* proposed an effective method for predicting the number of bugs that will be fixed in the future and the time required and *quickness*, a binary classification (slow or quick) of the time required to fix a bug [24]. In this line Ginger *et al.* investigated the relationship between bug report attributes in three FOSS projects –Eclipse, Mozilla, and GNOME– to build a prediction model. Their findings show that the reporter, the assignee and the date the bug report was opened were the attributes with the strongest influence on the fix-time of the bugs [6].

Some authors have decided to study how the experience of the author affects in the bug fixing activity. Development experience has been measured in several ways: number of commits [4], fixing activity [1] and ownership [5]. Thus, based on these measures, Izquierdo *et al.* analyzed some Mozilla modules expecting statistical differences between developers with different levels of experience and the introduction of bugs, but the results did not show correlation, meaning that more experience does not imply less bug introducing changes [10]. Also, Izquierdo *et al.* studied whether developers who introduced a bug were the same who fixed it; their findings show that in most cases people are usually fixing bugs that were introduced by other developers [9].

Other studies have analyzed the experience of the author to predict the future bugs, and although there is research literature that suggests that by using this information the prediction of bugs does not improve [22], others report that the change's defect probability decreases with higher experience of the developers [15]. However, it seems there is evidence that defect injection rates vary among different developers [13].

While the prior studies were built under the assumption that the previous modifications to a fixed line was the cause of the bug, the contribution of our study resides in the fact that we do not follow this assumption, which is sometimes flawed. Instead, we identify the *real* modification that introduced the bug [18]. Once this modification is identified, we compute some of the metrics found in previous studies and compare our results to them.

III. Methodology

In our study, the data analyzed has been obtained from the source code management system, the issue tracking system and the code review system. We illustrate our methodology in Figure 1, where the input is a set of bug reports randomly extracted from the issue tracking system. The steps of our process are given by white boxes; the colored boxes give the sources that have been used to obtain the information required in each step.

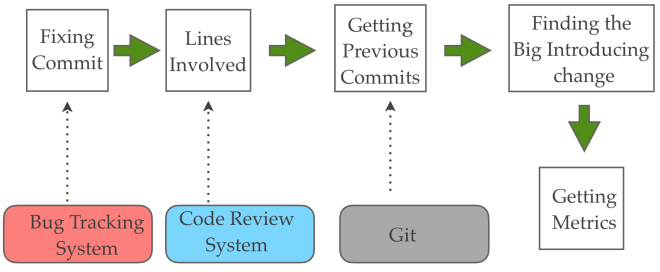

Fig. 1. The methodology used in this study, starting with the analysis of a fixing commit, determining the involved lines of code, obtaining the previous commits, finding the BIC and finally computing the value of the metrics for the whole process.

The following list is a detailed description of the steps:

1) Find the fixing commit of each bug report. The linkage of both elements was done manually since after fixing a bug, it is common practice in many open source projects that developers provide information, including a link to the commit in the versioning system, in the bug report in the issue tracking system.
2) Find the lines that this commit added, modified or deleted to solve the bug.
3) Obtain, for each of those lines, the commit that added, modified or deleted these lines previously. The result is a set with *previous commits* for each line.
4) Analyze which one of the previous commits was the BIC. In the case where the preceding commit is not the origin of the bug track back to a previous commit, until the line causing the failure is found. In the case where the fixing commit consist exclusively of new lines, analyze previous commits to nearby lines looking whether in any of them the developer *forgot* to add these lines. It should be noted that the SZZ algorithm discards fixing commits that only add lines.
5) Extract the date when the fixing commit, the BIC and the bug report have been submitted, as well as the author of each of the commits to calculate our metrics.

A. Metrics

To compute the metrics used in this study, we need to identify the BIC. Once this has been done, we are able to measure the time values. Next, we describe the metrics used in our study:

- *Experience until Bug Introducing Change* (EuBIC): Experience of the author of the BIC at the moment of doing the commit. Experience is measured in days from the first time that the author committed some code to the project until the BIC.
- *Time To Notify* (TTN): Time in days since the bug introducing commit was merged into the master branch until some developer notified the unexpected behavior and reported it in the bug tracking system.
- *Time To Fix* (TTF): Period in days from the notification of a bug report to when it was closed with a fixing commit.
- *Bug Fixing Time* (BFT): Time in days from the date of the BIC to the fixing commit date. This value can be also computed by adding TTN and TTF.

Figure 2 provides a visual explanation of the metrics proposed in this paper. For example, to obtain the metrics for bug report #13334 of ElasticSearch, we need to identify the fixing commit (which is $c6da8d5e1$) and the BIC ($c73fff7$). Then, analyzing the meta-data of theses commits, we obtain the date and the author for both of them: the bug report was done in September 4th 2015, the fixing commit in September 15th 2015, and the BIC was inserted in July 13th 2015 by a developer with almost 3 years of experience in the project. Hence, in our example, the values of the metrics under study are: 54 days for TTN, 11 days for TTF, 65 days for BFT and 1,097 days for EuBIC.

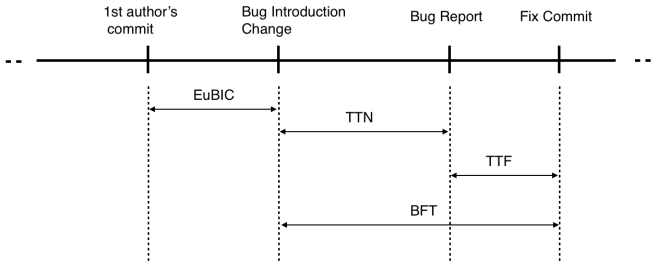

Fig. 2. Visualization of the periods under study: Experience Until Bug Introducing Change (EuBIC), Time To Notify (TTN), Time To Fix (TTF), Bug Fixing Time (BFT).

We would like to compare our metrics with the widely used SZZ algorithm. We will therefore assume that the closest commit in time (i.e., the previous one) of the SZZ algorithm is the one that induces the bug-fixing commit, as it has been done in previous works [4]. It should be noted, however, that the SZZ algorithm may not identify the BIC correctly, as its outcome could be a set of commits among which the *real* BIC is not included. In addition, in our comparison we discard those fixing commits with only new lines, since the SZZ algorithm removes them from the analysis.

IV. Evaluation

We have validated our methodology analyzing tickets from two projects, Nova and ElasticSearch, written in different programming languages.

Nova belongs to the OpenStack project, a cloud computing platform with a huge developer community (more than 5,000 developers) and significant industrial support from hundreds of organizations, among them several major IT companies such as Red Hat, Intel, IBM, HP, etc. The source code of Nova is written in Python and was particularly of interest because it is continuously evolving due to its very active community. Currently it has more than 44,000 commits with more than 2 million lines of code and around 1,000 contributors[1]. All its code is hosted and available in GitHub[2] and it is developed with git[3] as the source code management, Launchpad[4] as the issue tracking system and Gerrit[5] as the code review system.

ElasticSearch is a distributed FLOSS search and analytics engine written in Java. It has 26,000 commits and 764 contributors. Its labeling policy in the bug tracking system is very strict, thus we can be sure that tickets labeled as bug reports (i.e., the ones analyzed in this paper) are *real* bug reports. ElasticSearch hosts its code in GitHub[6]. The project uses GitHub's issue tracking and pull request system for bug tracking and reviewing. We use in this paper a data set from

[1]http://activity.openstack.org/dash/browser/repository.html?repository=nova.git&ds=scm
[2]https://github.com/openstack/nova
[3]https://git-scm.com/
[4]https://launchpad.net/
[5]https://www.gerritcodereview.com/
[6]https://github.com/elastic/elasticsearch/

TABLE I
MEAN IN DAYS OF TTN, TTF, BFT FOR THE NOVA PROJECT, BOTH FOR THE *real* BIC AS FOR THE BIC OBTAINED WITH SZZ

	TTN	TTF	BFT
Real	432	65	497
SZZ	260	52	312

TABLE II
MEAN IN DAYS OF TTN, TTF, BFT FOR THE ELASTICSEARCH PROJECT, BOTH FOR THE *real* BIC AS FOR THE BIC OBTAINED WITH SZZ.

	TTN	TTF	BFT
Real	312	14	326
SZZ	135	16	151

these two projects created in a prior research work, where we manually analyzed a total of 76 bug fixing commits of real bug reports in detail, looking in each one for the BIC that caused the failure [17].

To better understand how our results are related with the maintenance and evolution of a software, we have used the *OLS* (Ordinary Least Square) regression model to calculate the dependency of the different variables in the projects. We have used as well the *Pearson* method to compute the correlation between the variables.

V. RESULTS

We have calculated the values of TTN, EuBIC and TTF for 76 bug fixing commits, 39 belonging to Nova and 37 belonging to ElasticSearch. Table I and Table II show the means of Time To Notify (TTN), Time To Fix (TTF) and Bug Fixing Time (BFT) computed for the Nova and the ElasticSearch projects, respectively. The tables include the values for the *real* BIC (manually checked), and the ones that can be obtained by using the SZZ algorithm.

Figure 3 and Figure 4 show the values of TTN, TTF and EuBIC for both projects using box plots. We can see that it takes 158 days in ElasticSearch to notify 50% of the bugs, and 336 days for Nova. The mean is almost 200 days in ElasticSearch and around 350 days in Nova. TTF is low for both projects, but especially for ElasticSearch where there is almost no dispersion and bugs are fixed almost immediately after their notification. And Nova developers are slightly more experienced than ElasticSearch developers when they introduce bugs.

Figure 5 shows the correlation matrix between the metrics under analysis for the Nova project. A moderate negative correlation, -0.56, between the EuBIC of the author introducing the bug and TTN can be found. The value of this correlation for the ElasticSearch project is -0.383, which points to a weaker relationship[7].

Figure 6 shows scatterplots with the distribution of each pair of metrics for the Nova project. While we can infer a negative inclination in the relationship between TTN and EuBIC with

[7]Due to space constrains we do not show a similar figure for ElasticSearch.

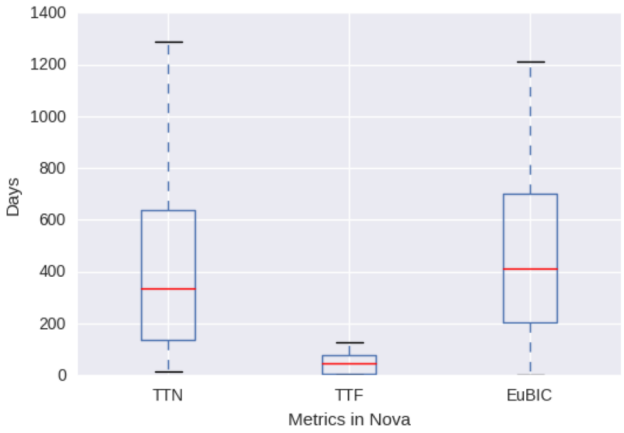

Fig. 3. Box-plots with TTN, TTF and EuBIC for the Nova project.

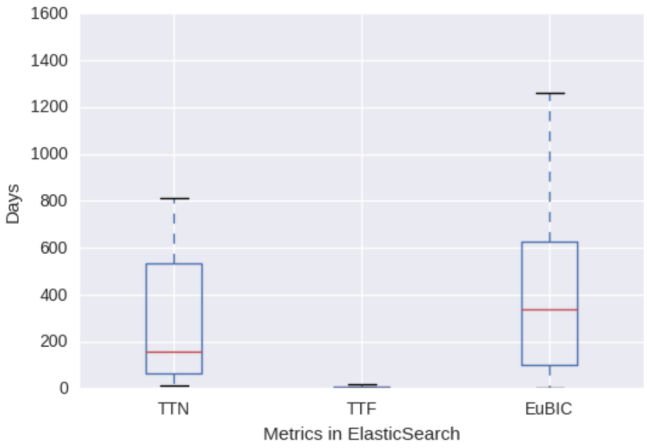

Fig. 4. Box-plots with TTN, TTF and EuBIC for the ElasticSearch project.

a R^2 value of 0.32, which indicates how closely these two variables are related, the plot for TTN or EuBIC against TTF does not indicate such tendency.

On the other hand, Figure 7 shows how the variables are related to each other in ElasticSearch. In this project, it is difficult to notice the how closely the variables are related, the value of the coefficient R^2 in each scatterplot is around 0.09.

To gain further insight into the problem, we have analyzed if the same person who introduced the BIC is also the one who reported and/or fixed it. Tables III and IV show the percentage of bugs in ElasticSearch and Nova where the same developer (1) notified and fixed the bug, (2) introduced the BIC and fixed the bug, (3) introduced the BIC and notified the bug, and (4) introduced the BIC, notified the bug and fixed it.

The highest percentage corresponds to the situation where the same developer who notifies the bug also fixes it. The second rank is for bugs that have been fixed by the same developer who introduced it, i.e., somebody else reported the bug; in the case of ElasticSearch, the percentage is as high as for the previous case. The cases where the same developer

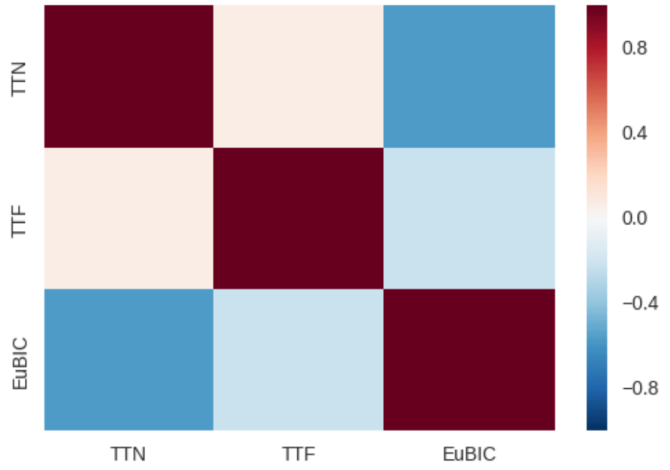

Fig. 5. Correlation Matrix of the variables measured for the Nova project. Correlation coefficients are colored according to the value; white means a correlation value of 0, the red tone increases with the grade of positive correlation while the blue tone decreases with the grade of negative correlation.

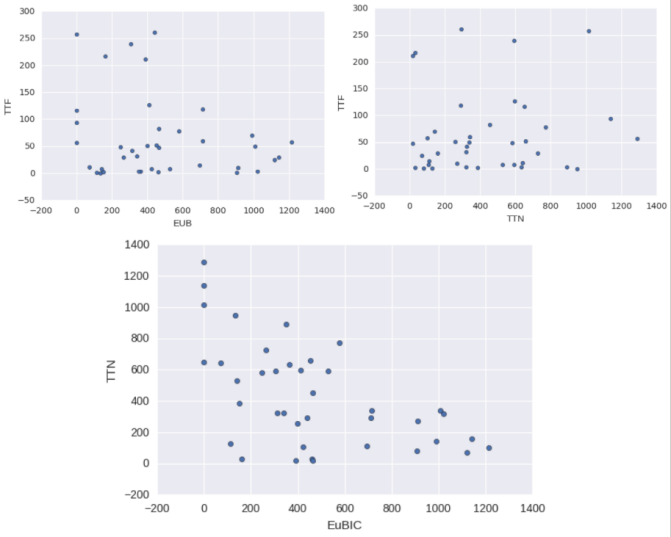

Fig. 6. Scatterplots with the metrics for the Nova project.

TABLE III
PERCENTAGE OF BUGS WHERE A DEVELOPER PERFORMED MORE THAN ONE OF THE TASKS UNDER STUDY (INTRODUCE, NOTIFY, FIX) FOR THE ELASTICSEARCH PROJECT.

	Notify-Fix	Introd.-Fix	Introd.-Notify	Introd.-Notify-Fix
Yes	16 (43%)	16 (43%)	11 (30%)	10 (27%)
No	21 (57%)	21 (57%)	26 (70%)	27 (73%)

introduced the bug and later on notified it are less frequent, but still happen often (30% for ElasticSearch and 13% for Nova). Finally, a bug introduced, notified and fixed by the same developer occurs not marginally (25% for ElasticSearch, 13% for Nova).

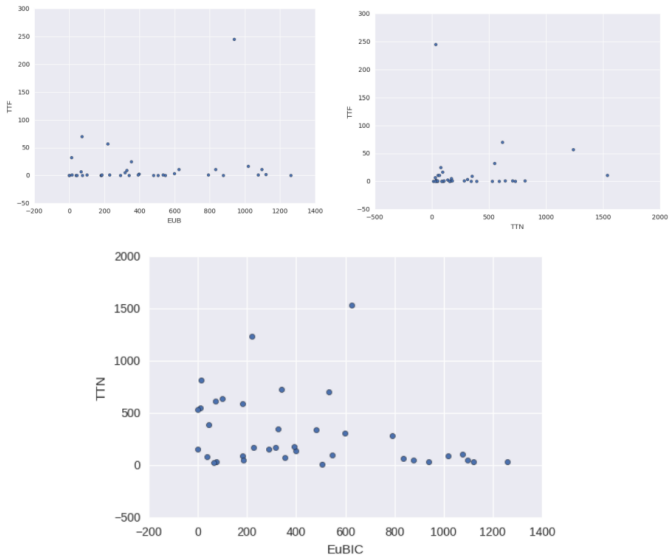

Fig. 7. Scatterplots with the metrics for the ES project.

TABLE IV
PERCENTAGE OF BUGS WHERE A DEVELOPER PERFORMED MORE THAN ONE OF THE TASKS UNDER STUDY (INTRODUCE, NOTIFY, FIX) FOR THE ELASTICSEARCH PROJECT.

	Notify-Fix	Introd.-Fix	Introd.-Notify	Introd.-Notify-Fix
Yes	30 (77%)	7 (18%)	5 (13%)	5 (13%)
No	9 (23%)	32 (82%)	34 (87%)	34 (87%)

VI. DISCUSSION

After analyzing several fixing commits in this study, we have seen that *real* (manually validated) TTN values differ substantially from the ones computed using the well-known and widely-used SZZ algorithm, identifying different BICs for 36% of the cases in Nova and for 24% in ElasticSearch. This is the reason why the mean TTN computed using SZZ is almost half the *real* one.

The median time to notify a bug is almost 200 days in ElasticSearch and about 350 days in Nova. If we want to calculate the median time of bug fixing for each project, we have to add the median TTF to the TTN, obtaining the BFT. The BFT in ElasticSearch does not differ much from the TTN, whereas in Nova it reaches 400 days. Kim and Whitehead in [11] reported values of BFT that range from 100 to 200 days. In our case studies, only ElasticSearch is in this range, although by very little margin. An explanation for the deviation is because Kim and Whitehead used SZZ to locate the BIC, which we have found is not only always *true*, but provides an optimistic value.

RQ1: The *real* value of the Time to Notify (TTN) a bug in both projects differs from the value calculated using SZZ.

We have studied the relationship between the variables under study. We have seen for Nova that TTN shows a tendency to increase for lower values of EuBIC, meaning

that one of the reasons for high values of TTN could be that the developer who introduced the bug did not have much experience on the project at that moment.

RQ2: The experience of the author who introduced the bug in the source code and the Time To Notify (TTN) are (negatively) correlated. On the contrary, the results do not show correlation between the Time To Fix (TTF) with the Time To Notify (TTN), nor with the experience of the author (EuBIC) who introduced the bug in the source code.

Given that we have the meta-data from the source code management systems, we could trace how many developers do several tasks in the bug introduction $->$ notification $->$ fixing process. The results show that notifiers often also fix the bugs. Developers who introduced the bug also appear to fix it (after a third party notifies it) and sometimes they even notify the bug without providing a solution. Interestingly enough, we have found that the number of times where a developer is involved in the three tasks for the same bug report is not negligible. These results raise a lot of questions that are very interesting from the community perspective. One possible reason that explains the low percentage of authors who fix their bugs may be because they are no longer in the project when the bug is reported. We have looked for this and have found that the number of authors who have abandoned the project when the bug was reported was only one in ElasticSearch and two in Nova. This means that other reasons have to be more important than code ownership when it comes to a bug. On the contrary, the high percentages of developers notifying and fixing bugs may be because when they reporting the bug, they already know how to fix it, and this might have and impact reducing the time to fix a bug.

All in all, we think that the Time To Notify a bug can demonstrate to be a good metric of a project, as it provides good perspective on how the project is maintained. Other metrics can be easily circumvented, as for instance the number of commits or of mailing list messages, and may result in perverse effects if they are linked with incentives or productivity measures. For TTN (and BFT) this is more difficult to achieve, but not impossible: one could think of a developer introducing a bug and the notifying and fixing it shortly thereafter to obtain lower values.

Given that one of the problems with TTN (and the rest of associated metrics) is that finding the BIC is difficult and has to be done in a manual way if high precision is the goal, we would like to propose that bug tracking systems include an additional field when closing a bug. In this field, the fixing developer could specify which one was the BIC. It should be noted that there is no one more qualified at this moment that knows where and when the bug was caused. That way the project could have an accurate measure of its maintenance activities and processes, and researchers will access better data and build a better models that estimate effort and the total cost of the software [2], [16], [23].

VII. Threats to validity

The major threat to the validity of our study comes from the limited sample size of tickets used in this research. It is, however, a relatively high number of tickets considering that the analysis was done manually and in detail to reach reliable results, but there is a long way to get a representative sample from a large variety of diverse Free/Open Source Systems, or even other software projects. Our analysis requires a lot of human effort, so meaningfully increasing the number of tickets is human resource intensive and difficult. However, it should be noted that our numbers are in the order of magnitude of similar studies: so, for instance, Hindle's *et al.* article on large commits considered 100 commits in total [8].

Other internal threats to validity are:

- We are only using part of the information that the tickets provide, like comments and text. There could be some patterns that could potentially be found that could help in identifying bug fixes more accurately. However, we think that this information is enough to determine where the bug introducing change is, as comments found in the ticket usually give enough information about what is failing, and this can be tracked back to the code.
- In some cases, when only new code was added in the bug fixing commit, researchers may have problems to compare their TTN with the one provided by SZZ, because of SZZ does not considered new additions in the fixing commits.
- The experience of the developer calculated from its first commit in the system may not be the best definition to calculate his/her experience; an early, but low activity contributor could potentially have less experience in the project that a new, but very active contributor.

The most important external threats are:

- OpenStack is a special project with a very rapid evolution, and a very active community of developer and ElasticSearch is relatively new project with a strong criteria in the bug fix activity. In other projects, with less commits per year, results may be totally different.
- The programming languages analyzed in this research are Python and Java: It may happen that other programming languages present different results.

VIII. Conclusion and further research

The study we have performed on Nova and ElasticSearch has shown that the mean Time To Notify (TTN) a bug is over ten months, whereas using the SZZ algorithm to calculate this metric we obtain a value slightly higher than five months.

Our study shows a negative correlation between the TTN and EuBIC, which means that the time to notify an error is related to the overall experience of the author.

In addition, we propose to build a new field/option in the bug tracking system to record which commit was the BIC, increasing that way the confidence of which previous commit inserted the bug.

Once we have found this, it makes sense to explore, as future work, to which extent this happens in other projects with probably higher number of tickets and with a higher number metrics.

Another future line could be to perform a detailed research on how this metric could be helpful in defect prediction field, building new models or improving them.

The full automation of the methodology used in this paper is also interesting from a practical point of view. That would provide software projects with a valuable tool for understanding how bugs are introduced, and therefore calculate these metrics for mitigation.

Replication package: we have set up a replication package[8] including data sources, intermediate data and scripts.

ACKNOWLEDGMENTS

We want to express our gratitude to Bitergia[9] for their open source tools to mine the repositories of the projects, and the support they have provided when questions have arisen. Also, we acknowledge the Spanish Government, through project TIN2014-59400-R. The last two authors are funded in part as well by the Region of Madrid under project "eMadrid: Investigación y Desarrollo de tecnologías educativas en la Comunidad de Madrid" (S2013/ICE-2715).

REFERENCES

[1] S. N. Ahsan, M. T. Afzal, S. Zaman, C. Guetl, and F. Wotawa. Mining effort data from the oss repository of developers bug fix activity. *Journal of IT in Asia*, 3:67–80, 2010.

[2] J. Asundi. The need for effort estimation models for open source software projects. In *Proceedings of the Fifth Workshop on Open Source Software Engineering*. ACM, 2005.

[3] P. Bhattacharya and I. Neamtiu. Bug-fix time prediction models: can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 207–210. ACM, 2011.

[4] J. Eyolfson, L. Tan, and P. Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162. ACM, 2011.

[5] D. M. German. Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):367–384, 2004.

[6] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[7] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 495–504. IEEE, 2010.

[8] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108. ACM, 2008.

[9] D. Izquierdo-Cortazar, A. Capiluppi, and J. M. Gonzalez-Barahona. Are developers fixing their own bugs?: Tracing bug-fixing and bug-seeding committers. *International Journal of Open Source Software and Processes (IJOSSP)*, 3(2):23–42, 2011.

[10] D. Izquierdo-Cortázar, G. Robles, and J. M. González-Barahona. Do more experienced developers introduce fewer bugs? In *IFIP International Conference on Open Source Systems*, pages 268–273. Springer, 2012.

[11] 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*, pages 173–174. ACM, 2006.

[12] L. Marks, Y. Zou, and A. E. Hassan. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 11. ACM, 2011.

[13] S. Matsumoto, Y. Kamei, A. Monden, K.-i. Matsumoto, and M. Nakamura. An analysis of developer metrics for fault prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, page 18. ACM, 2010.

[14] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[15] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[16] G. Robles, J. M. González-Barahona, C. Cervigón, A. Capiluppi, and D. Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 222–231. ACM, 2014.

[17] G. Rodriguez. Analysing on how the bugs are injected into the source code. In *Proceedings of the Doctoral Consortium at the 12th International Conference on Open Source Systems, Gothenburg, Sweden, 30 May, 2016*. University of Skövde, 2016.

[18] G. Rodriguez-Perez, J. M. Gonzalez-Barahona, G. Robles, D. Dalipaj, and N. Sekitoleko. Bugtracking: A tool to assist in the identification of bug reports. In *IFIP International Conference on Open Source Systems*, pages 192–198. Springer, 2016.

[19] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *Proceedings of the 2005 International Workshop on Mining software repositories*, pages 1–5, 2005.

[20] G. Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project*, 7007(011), 2002.

[21] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1. IEEE Computer Society, 2007.

[22] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Programmer-based fault prediction. In *PROMISE10: Proceedings of the sixth international conference on predictor models in software engineering*, page 19, 2010.

[23] H. Wu, L. Shi, C. Chen, Q. Wang, and B. Boehm. Maintenance effort estimation for open source software: A systematic literature review. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, pages 32–43. IEEE, 2016.

[24] H. Zhang, L. Gong, and S. Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1042–1051. IEEE Press, 2013.

[8] http://gemarodri.github.io/Reprodu-Package/

[9] http://bitergia.com/