Abstract—Businesses and research establishments are increasingly turning to Free and Open Source software (FOSS) as a means to lower software development, acquisition, and deployment costs. However, software quality and security remains key stumbling blocks to full scale FOSS adoption and deployment. Yet improvement in the quality and security of FOSS depends on the rate at which a community of volunteers report and fix bugs. The aim of this research is to understand the community governance of the bug reporting and fixing process. We link data obtained from bug tracking systems, source code repositories, and mailing lists and applied various metrics to investigate the dynamics of bug communities in 285 projects. The results of our study show that the identity of bug reporters or fixers, the size of the bug community and code are key factors in ensuring quality software. The implications of these findings for bugs governance, software and code quality, empirical research difficulties, and future research directions are also discussed.

Index Terms—Open source software projects, open source communities, software bugs, software quality.

I. INTRODUCTION

A multitude of interrelated factors are contributing to the upward trend in global adoption and utilization of Free and Open Source Software (FOSS) [1]. Some of these factors include: an alternative Bazaar style of developing software which harnesses diverse talents of globally distributed teams of developers, freedom from vendor lock-in, lower total cost of ownership and hybrid business models opportunities, and learning and knowledge sharing prospects [2]. However, even though there is continued improvement in the quality of FOSS [3], the adoption and integration of FOSS technologies and services into the operation of businesses is largely hampered because many users have little confidence and trust in the quality and security of FOSS. Since the wish of every software user is to have a reliable application which is free of bugs, then the presence or lack of bugs is one among many measures that can help us determine the quality and security level of a given piece of software.

A bug is an undesirable companion of any software, and the process of debugging has a crucial role in ensuring that the number of bugs is kept to minimum. Bugs and debugging are an integral part of FOSS development and many projects are hailed for the rate at which volunteers contribute to this process. It is argued that the iterative nature of this process leads to the evolution, improved quality, and reliability [3], [4] of the software. The importance of finding, reporting, and fixing bugs in FOSS is well captured in Linus’ law [5], which states,”Given enough eyeballs, all bugs are shallow”. The law assumes that for any bazaar-style project with large enough testers or bug reporters and fixers, almost every problem or bug in the software will be spotted and fixed quickly.

A lot of research exist which sheds light on the FOSS bug reporting and fixing process. However, most research concentrates on a limited numbers of projects. We posit that data from a single repository (e.g. bug tracking systems alone) from one or two projects will most likely not give a comprehensive picture of the dynamics of the debugging process. As pointed out by Zhenmin Li, et al. [6], “small number of bugs [and projects] may lead to non-representative results”. Furthermore, depending on the project’s technical infrastructure, bugs may be found in any repository, or even as attachments to code snippets [7]. This makes it difficult not only for researchers intending to link data from multiple repositories but also for quality assurance teams to track both bugs and people involved in bug reporting and fixing.

In this research, we obtained dumps and link data from three repositories (bug tracking systems, source code management systems (SVN), and mailing lists) maintained by 285 FOSS projects in the FLOSSMetrics database [8]. Using various metrics we analyze the community structure governing the bug reporting and fixing process. In this structure, we expect the identity of individuals reporting and fixing bugs, and the status of bugs are all related and will affect, in some way, the dynamics of the bug reporting and fixing process. Furthermore, we hypothesize that as the code base of a project grows, so is the chance of the software becoming buggy. The bug reporting and fixing activity may also increase exponentially. Thus, we are also interested in investigating whether there is any significant relationship between project size in terms of Source Lines of Code (SLOC) and the number of bugs reported and fixed.

The rest of the paper is structured as follows. In section 2 we provide the background and work related to our research. Our research methodology and data sources are presented in section 3. Our data analysis and results are discussed in section 4. We conclude our research and present our future work in section 5.
II. BACKGROUND AND RELATED WORK

The bug reporting and fixing process proves to be a dynamic and complicated process involving both software developers and users. Understanding the complex interplay between bugs and community of volunteers may help us better manage and allocate projects’ human resources, gauge the quality of the software, improve bug triage, and design new and improve existing tools for bug detection [9], [10] and reporting.

Furthermore, this research will provide insight into the FOSS development process by helping us understand who is reporting what types of bugs, who is communicating with whom and is fixing which bug. FOSS project managers and quality assurance teams may find the results of this research useful in helping them understand and plan software debugging issues. For FOSS users and businesses, such findings may increase their confidence and trust in the quality and security of FOSS.

Many researchers contributed to FOSS body of knowledge by investigating the relative time it takes to fix bugs [11], [12], characterizing bugs according to types of errors [13], classifying defect-prone files [14], helping us understand the role of developers in the bug fixing process [15], proposing ways of coping cope with problems associated with opening up bug repositories [16], and investigating the structure and the coordination practices adopted by development teams during the bug-fixing process. However, while each project is unique, data from one or two projects will most likely not give us a comprehensive picture of the complex bug reporting and fixing process. A review of the research literature on FOSS bugs (Table I) shows that researchers, with the exception of [15], study one or two, at most, nine projects [17]. Thus, a relatively large sample of projects may yield an added dimension that would have been difficult to observe from a small number of projects. Furthermore, researchers need to leverage the wealth of bug information available across repositories and apply cyber-archaeology to help them link bugs data available not only in bug databases, but also in source code repositories and mailing lists.

Many researchers contributed to FOSS body of knowledge by investigating the relative time it takes to fix bugs [11], [12], characterizing bugs according to types of errors [13], classifying defect-prone files [14], helping us understand the role of developers in the bug fixing process [15], proposing ways of coping with problems associated with opening up bug repositories [16], and investigating the structure and the coordination practices adopted by development teams during the bug-fixing process. However, while each project is unique, data from one or two projects will most likely not give us a comprehensive picture of the complex bug reporting and fixing process. A review of the research literature on FOSS bugs (Table I) shows that researchers, with the exception of [15], study one or two, at most, nine projects [17]. Thus, a relatively large sample of projects may yield an added dimension that would have been difficult to observe from a small number of projects. Furthermore, researchers need to leverage the wealth of bug information available across repositories and apply cyber-archaeology to help them link bugs data available not only in bug databases, but also in source code repositories and mailing lists.

<table>
<thead>
<tr>
<th>Projects studied</th>
<th>Bug tracking tool</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozilla/300 projects</td>
<td>Bugzilla</td>
<td>Cause of bug, software component affected.</td>
</tr>
<tr>
<td>Apache</td>
<td>JIRA, SourceForge, Bugzilla</td>
<td>Performance characteristics of the bug fixing process.</td>
</tr>
<tr>
<td>Nine projects</td>
<td>Bugs Syss</td>
<td>Different bug fixing regimes.</td>
</tr>
<tr>
<td>Mozilla/300 projects</td>
<td>Bugzilla</td>
<td>Effort spent fixing bugs.</td>
</tr>
<tr>
<td>Firefox [18]</td>
<td>Bugzilla</td>
<td>What is stored in and how bug repositories are being used.</td>
</tr>
<tr>
<td>Eclipse, Firefox</td>
<td>Bugzilla</td>
<td>Predict the time to fix a bug.</td>
</tr>
<tr>
<td>Eclipse/300 projects</td>
<td>Bugzilla/SourceForge issue tracker</td>
<td>Understand the role of core developers in FOSS development.</td>
</tr>
</tbody>
</table>

MySQL queries paths performed to obtain the metadata required for our analysis. The primary data comes from the FLOSSMetrics project database. FLOSSMetrics uses tools to analyze and maintain three public repositories; source code management systems (SCM), mailing lists, and bug tracking systems (BTS). For each repository, MySQL dumps were downloaded and restored into a local database. The metrics in Table II were then composed from each datasource and stored as n-tuples in four MySQL tables. Each table was queried and the results compared to provide the data needed for our analysis. From the FLOSSMetrics database, 285 projects with a complete set of data from the three repositories were selected for our study.

![Fig. 1. Outline of research methodology](image)

TABLE II: METRICS DEFINITION

<table>
<thead>
<tr>
<th>Metric name</th>
<th>Description</th>
<th>How calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_bugs</td>
<td>Number of bugs per project.</td>
<td>From bugs table, for each project, count the number of bugs reported.</td>
</tr>
<tr>
<td>N_submitters</td>
<td>The number of bug reporters per project.</td>
<td>From changes table, for each project, count the number of bug reporters.</td>
</tr>
<tr>
<td>SLOC</td>
<td>Total source lines of code in a project.</td>
<td>Select SLOC count for each project in the metrics table.</td>
</tr>
<tr>
<td>nposters</td>
<td>Total number of mailing list posters in a project.</td>
<td>From ml table, count all posters in the mailing list of each project.</td>
</tr>
</tbody>
</table>

The bugs table contains general information about bugs (bugID, date submitted, status, priority, assigned to whom, submitted by who). The changes table contains information on the changes a unique bug (bugID) underwent during its life cycle. For example, which field in the bugs table about this bug has changed; what the old value was; the date the changes were made; and who submitted those changes. The SCM metrics table contains commits and committers as well LOC and SLOC counts. For the mailing lists data, for each of the 285 projects, *.sql files dumps were downloaded and stored into the mailing list (ml) table. The methodology described in [2] was used to list the total number of posters for each project.

IV. RESULTS AND DISCUSSION

A. Exploratory Study

Few studies exist which attempts to link data from multiple
FOSS repositories of this nature. Thus, we begin our data analysis with an exploratory study that leverages the conceptual framework shown in Fig. 2. An exploratory study, according to [20], is undertaken when not much information is available on how similar research issues have been solved in the past.

Fig. 2. Conceptual research framework for analyzing the dynamics of bugs in Foss projects.

The framework in Fig. 2 presupposes bugs can be found in all of the three repositories. A "bug community" is composed of bug reporters and fixers and may involve the participation, at varying degree, of passive and active users, and developers and co-developers. As such, our analysis explores bug reporting and fixing from the perspective of three communities in bug tracking systems, source code versioning systems (CVS/SVN), and mailing lists. These communities and their activities are analyzed in this research. It is hoped that our research results can be feedback to FOSS projects and communities to improve resource management and allocation, offer insight into software and code quality, aid quality assurance teams in bug triage, help improve the design of debugging tools, among others.

| TABLE III: DESCRIPTIVE STATISTICS OF METRICS |
|-------------------|-----------------|--------------------|-------------------|
| Identity          | Mean            | Median            | Std. Dev.         | Max.            | Sum    | % in proj |
| N_bugs            | 330.82          | 2.97              | 13.23             | 1128243.52      |        |          |
| N_submitters      | 110.00          | 2.00              | 2.00              | 89678.00        |        |          |
| nposters          | 559.075         | 3.585             | 109.703           | 8295727.602     |        |          |
| SLOC              | 14.533          | 35.264            | 3486.792          | 209.490        |        |          |
| Kurtosis          | 15.706          | 35.264            | 3486.792          | 209.490        |        |          |
| Maximum           | 4096            | 69                | 12699             | 130174261      |        |          |
| N valid           | 94285           | 47653             | 996694            | 321549402      |        |          |

As shown in Table III, 996694 (mean=13.23) mailing lists posters and 47653 (mean=2.97; Std. Deviation = 3.585) bug reporters contributed a total sum of 94284 bugs (N_bugs) to the 285 projects. The mean bug per project was 330.82 (Std. Deviation = 559.075). The projects average over 1128 KSLOC per project (Std. Deviation = 89678.00). The box plots in Fig. 3 compare the distribution and skewness of the data in each metric. In comparison, it can be seen that the SLOC and mailing lists data are much skewed, with extreme values and outliers.

B. The Topology of Bug Communities

A lot of effort is invested in helping improve the management, reporting, and resolution of bugs in FOSS projects. Given that software is prone to bugs, perhaps we can reduce the bugginess and speed up the debugging process by understanding the way bug communities in various projects work. Bug reports usually have the “identity” of the persons involved in bug triage; who submitted the bug, to whom the bug is assigned to, and who fixes the bug. In defining the topology of bug communities, we identified five groups of people and computed their percentage presence in the projects (Table IV):

- **Group 1**: Non anonymous bug submitters whose bugs are assigned to non anonymous individuals;
- **Group 2**: Anonymous submitters;
- **Group 3**: Anonymous individuals whose bugs are closed;
- **Group 4**: Anonymous bug submitters whose bugs are assigned to anonymous individuals;
- **Group 5**: Anonymous individuals whose bugs get deleted.

As shown in Equation 1, the almost mirror image (P) of some of the groups. In 78.25% (N=223) of the projects non anonymous bug submitters (known identities) have their bugs assigned to non anonymous (Group 1).

\[
P : \left( \text{Group}_1 \right) \mapsto \left( \text{Group}_4 \right) \text{.} \tag{1}\]

The mean number of non anonymous bug submitters whose bugs are assigned to non anonymous individuals is 168.56 (Std. Dev. = 334.200). Furthermore, over 92% of those assigned bugs in this group were found to belong to members listed in their respective project’s team. We suspect that there is some kind of “bugs trading” between the core or
C. Projects Size (in terms of SLOC) vs. Number of bugs

In investigating the relationship between projects SLOC size and the number of bugs, we mapped each project’s SLOC data with its corresponding bug’s data. The outcome shows a significant correlation, with Pearson $r = 0.636$, and Kendall’s $t_{au,b} = 0.998$ for the ranked values (for both values, $p = 0.01$). Furthermore, curve estimation regression statistics was applied to model the relationship between the two variables. Fig. 5 shows the regression, model fit, and linearity equation for $\ln(SLOC)$ (transformed).

A linear and quadratic model explains 90% ($R^2=0.900$) and 95.2% ($R^2=0.952$), respectively, of the variability. Although $R^2$ for the quadratic model is larger, it is not clear whether this is due to the model dependence on the extreme large SLOC values or on chance with extra parameters in the model fit. These results show that with increase growth in the code base, the chance of the software becoming buggy also increases.

![Fig. 4. Relationship between number of bugs and mailing lists participants (both axis in log scale).](image)

![Fig. 5. Linear and quadratic regression models showing the relationship between in SLOC and number of bugs.](image)

V. CONCLUSION

In this paper we have presented a methodology which utilized data from multiple repositories (bug tracking, CVS/SVN, mailing lists) and a conceptual framework to investigate and map out communities involved in the bug reporting and fixing process in 285 FOSS projects. Using various metrics and statistical measures, we discussed the topology of bug communities by revealing the identities (anonymous and non anonymous) people assume in the bug reporting and fixing process. Furthermore, we grouped people in bug trackers and quantified their contribution to the debugging process. Mailing lists community size was found to correlate significantly with the number of bugs in a project ($r = 0.790$). We established, with 90% certainty ($R^2=0.900$), that project size in terms of SLOC is highly correlated ($r = 0.636$) with the number of bugs.

While this study is not without its limitations, the results may serve as an entry point in helping us understand the dynamics of bugs’ communities in FOSS projects. Software testing and debugging tools play a fundamental role in software development. However, we believe that understanding, nurturing, and supporting the activities of bug communities is an essential endeavor project managers or module maintainers must undertake to ensure quality software is delivered to users. In this regard, our research may serve as the starting point in helping us understand the implications bug communities have for software and code quality.

VI. FUTURE WORK

First, there are few large projects in our sample; some are one-person endeavours. Therefore, we plan to repeat this study with a large FOSS project (e.g. Apache) to see if the finding in this research can be generalized. Second, we are using this research as a base to investigate “bug networks” in which two or more projects (say $i$ and $j$) are linked by a bug reporter ($r$) if s/he reports $k$ bugs in both projects. Then we can compute $\lambda = k^* i + k^* j + \ldots kn$, to give us the number of
bugs \( r \) contributes to projects \( i \) and \( j \). Such a network will enable us identify star or linchpin bug contributors and add value to our understanding of the dynamics of bug communities.

**ACKNOWLEDGMENT**

We are grateful to the FLOSSMetrics consortium for providing us the data used in this research. We are greatly indebted to the 3rd IEEE International Conference on Information Management and Engineering (IEEE ICIME 2011) anonymous reviewers for their constructive comments and suggestions which helped us improve the quality of the earlier version of this manuscript.

**REFERENCES**


