Got Issues? Who Cares About It?
A Large Scale Investigation of Issue Trackers from GitHub

Tegawendé F. Bissyandé[1], David Lo[2], Lingxiao Jiang[2], Laurent Réveillère[3], Jacques Klein[1] and Yves Le Traon[1]
[1]SnT, University of Luxembourg, Luxembourg
[2]Singapore Management University, Singapore
[3]LaBRI, University of Bordeaux, France
tegawende.bissyande@uni.lu, {davidlo, lxjiang}@smu.edu.sg, reveille@labri.fr, {jacques.klein, yves.letraon}@uni.lu

***Abstract*—Feedback from software users constitutes a vital part in the evolution of software projects. By filing issue reports, users help identify and fix bugs, document software code, and enhance the software via feature requests. Many studies have explored issue reports, proposed approaches to enable the submission of higher-quality reports, and presented techniques to sort, categorize and leverage issues for software engineering needs.**

Who, however, cares about filing issues? What kind of issues are reported in issue trackers? What kind of correlation exist between issue reporting and the success of software projects? In this study, we address the need for answering such questions by performing an empirical study on a hundred thousands of open source projects. After filtering relevant trackers, the study used about 20,000 projects. We investigate and answer various research questions on the popularity and impact of issue trackers.

I. Introduction

Developers and users often find issues with software systems and are encouraged to report them in the available issue trackers that are set up by development teams. There exists a variety of issue tracking systems. Popular systems include Bugzilla, Jira, etc. Other development platforms such as Google Code, GitHub or Freecode have in-house implementations of issue tracking systems. Developers thus have ample opportunities to use issue tracking systems in their development and maintenance process.

Many research studies have investigated ways to improve the user experience in issue tracking systems [4], [11], while others have proposed techniques for helping developers manage and address issue reports [1], [24]–[26]. Also, a number of approaches have linked issue reports to changes in source code [5], [30]. Despite the numerous studies that propose techniques for improving the issue reporting process, for enhancing the management of issues, and for leveraging issues in other software engineering tools, there have been limited surveys that investigate the actual adoption of issue trackers in software development and the impact of their adoption to project success.

Our goal in this paper is to fill a gap in the research relevant to software issues by investigating the adoption of issue trackers in software projects. Such a study could *make practitionners aware of what to expect* when they setup issue trackers for their projects, and would *make researchers aware of the scope and potential bias* of issue trackers data. In this work, we consider software projects where issue trackers are systematically set up and where there is little barrier to their usage. We also investigate the impact of the usage of issue trackers on the ultimate goal of project success. Project success can be characterized in diverse dimensions: monetary returns, number of downloads, or popularity. This study focuses on open source projects, and we only consider popularity as a measure of project success.

Our study exploits data from GitHub, a super-repository of software projects containing millions of projects. GitHub is free for open source projects and implements an in-house issue tracking system where users can file issues and tag them into self-defined categories. The issue tracking system is easy to use and is systematically provided to all projects hosted in GitHub. We collect a hundred thousands of projects from GitHub and investigate the adoption of issue trackers as well as the impact of issue tracking on software success.

Our contributions are as follows:

1) We perform a large scale study on thousands of software projects.To the best of our knowledge, it is the largest study conducted on issue trackers.
2) We investigate the adoption of issue trackers in terms of the projects that utilize them, the people that report issues, and the kind of issues that are usually reported.
3) We discuss our findings and share some insights on the correlations between the quantities of issue reports and various characteristics of software projects.
4) We also compare our findings with a previous study [27] on Firefox development where the authors have found that the size of the user community influenced the time-to-fix rate of bugs. We find that in general this correlation is small.

The structure of this paper is as follows. In Section II, we introduce GitHub and its issue tracker. Next, we elaborate our empirical study methodology in Section III. Section IV presents the findings of our empirical study; it highlights a number of research questions and their answers. We discuss threats to validity in Section V. Section VI describes related work. We conclude and mention future work in Section VII.

II. Preliminaries & Problem Definition

In this section, we discuss the importance of issue tracking systems in software development, and briefly describe GitHub, the source platform of the projects in our dataset.

978-1-4799-2366-3/13/$31.00 ©2013 IEEE

A. Issue trackers

Software development generally produces programs with two caveats: (1) they are often incomplete with respect to certain features, and (2) they are usually buggy. Developers testing the programs and end-users using the programs may then need to report *issues* when they believe that the product is not performing as it should or that it could be better at what it does. The valuable feedback from the user community is actually widely accepted as vital information for any software development [4]. In practice, an issue report is a request for improving a software system, fixing a bug, adding new features, or enhancing documentation. Project development teams may also use issues to flag "TODO" tasks.

Issue tracking systems, also known as issue trackers, are software products that aim at facilitating the management of issues in a software development project, by providing a feature-rich interface to ease issue reporting by the user community. It also enables a better follow-up of user's concerns by project developers. Studies have shown that issue trackers can be beneficial to both open and closed source projects [9], when they are publicly available for use.

There is a plethora of issue trackers used in software development projects. A number of them are more commonly referred to as bug trackers as they focus more on bug reports. Many development teams, including those at the Apache Software Foundation or the Linux kernel community, rely on standalone products such as Bugzilla[1], and JIRA[2] to track defects in their code, plan fixes, etc. Project hosting platforms, on the other hand, often resort to in-house implementations of issue trackers that are designed specifically to include a set of features in line with their hosting capabilities. GitHub belongs to this second category and associates with each hosted project an issue tracker that provides the usual facilities in issue tracking, such as filing issue tickets, tagging them accordingly to the nature of the issue, and labeling them as the state of resolution evolves. A project development team on GitHub can choose to enable or disable the in-house issue tracker. A team can also choose to use GitHub as a mirror for their project, while hosting their main development and maintenance activities somewhere else.

B. GitHub

The GitHub project hosting platform was launched in 2008 and has since grown to be one of the premier sites where over 3,000,000 projects are hosted and managed. The success of GitHub is largely attributed to the concept of *social coding* which has created a developer-friendly environment, allowing developers to network, collaborate and promote their projects. It enables *forking*, which allows to create copies of repositories, and *watching*, which allows developers to register to the events in a given project. It also supports *following*, which allows developers to subscribe to the activites of one another.

GitHub provides an extensive set of REST [8] APIs[3] that we can use to retrieve information on many project repositories whose contents are publicly accessible.

[1] http://bugzilla.org
[2] http://atlassian.com/software/jira
[3] http://developer.github.com

The projects hosted by GitHub offer the kind of diversity that is appealing for large empirical studies. Indeed, projects in GitHub produce software from various application domains, from web applets, gaming software, to operating systems. These projects are conducted by teams of developers of varying sizes—between 1 and several thousands, and include source code from a myriad of programming languages.

While our study focuses on the issue trackers managed by the different projects, there is a need to correlate statistical findings on issues with different characteristics of the projects. Thus, we are required to (1) collect a huge amount of data from GitHub and (2) infer important information, such as a project's number of lines of code, that is not directly available through the API, making a large study with GitHub challenging.

Other project hosting platforms, including SourceForge, GoogleCode and Freecode, also provide issue trackers to hosted projects. In this paper we focus on GitHub, but we plan to investigate those issues trackers in future work, to compare the findings, and draw new insights.

C. This study

A large body of literature has discussed the importance of issue trackers in the context of software development [15], [22], [27]. Other studies have also investigated user involvement in issue tracking based on issues reported in a few software projects [9]. A few other techniques, such as duplicate bug retrieval techniques [24], [25] and techniques to link code changes with issue reports [5], [30], have been proposed with the assumption of the prevalence of issues.

In this study, we consider a hundred thousands open source projects to investigate the popularity of issue trackers to provide insights on how issue trackers can be leveraged for improving software development.

III. Methodology

For our empirical study, we consider the first 100,000 projects returned by the GitHub API following a search request on available project repositories. These projects were retrieved with GitHub API v2 which returned a random list of projects[4].

We collect our dataset based on information from those projects. The obtained dataset is furthermore curated to distinguish projects that can introduce bias in our analysis from others: e.g., some projects are hosted in GitHub but maintain their issue tracking system outside of GitHub. Last but not least, we statistically characterize the popularity and usage of issue trackers in several dimensions, relying on well-known metrics to assess the statistical significance of our findings.

A. Collecting the dataset

For each project in our large set of projects from GitHub, we collect data related to a number of important software artefacts that commonly occur in development processes and are leveraged in this study.

[4] The list is available at http://momentum.labri.fr/orion/project_list.txt

a) Project information: The GitHub social coding site provides a number of features that enable developers to monitor and track activities in a project repository. In this study, we focus on two concepts, namely *watchers* and *forks*, that can be used as metrics for evaluating the popularity and thus the success of a project. The "watchers" metric gives an indication of the amount of attention that is given to a project by the developer community. Developers who subscribe as watchers to a project typically use, report bugs and incidentally promote the project in their developer social network. Similarly, "Forks" is a useful metric for measuring the active involvement of the developer community in the growth of a project's code base and the improvement of its quality. Indeed, developers often create copies (i.e., fork) of project code bases for continuing the development in parallel, while merging their improvements from time to time with the mainline repository. Though these metrics are not absolute, they provide good insights on the popularity of a project.

b) Issue reports: For each project, we also collect all (open and closed) issues reported through its tracker. For each issue, we collect information related to the reporter's identity, the labels used by the reporter to categorize the issue, etc.

c) Lines of code: We consider the number of lines of code (LOC) in each project. Because GitHub uses the git[5] software configuration management system (SCM) to store software revisions, we download the actual git repository for the project, and rely on the SLOCCount[6] utility to compute the actual lines of code in the latest revision of the project, ignoring code comments and blank lines.

d) Developer contributions: Finally, we consider for our study the development team for each project. Because git records code contributors' names and email addresses with each revision in the repository, we are able to accurately identify all contributors to a project code base, whether they are registered to GitHub or not. Indeed, the git SCM distinguishes between revision authors, who are the end-contributors to the code base, and committers, who have access to the mainline repository and can forward contributions from revision authors.

B. Curating the dataset

GitHub is a hosting site that is used by different project teams to get exposure in the developer community. Thus long-lived projects whose active development has started before the era of GitHub keep managing their projects on their own development sites while mirroring their source code repository in GitHub. Examples of such projects include the numerous projects developed under the auspices of the Apache Software Foundation and the Linux kernel mainline tree. These projects disable the issue tracker provided by GitHub to avoid confusion on which tracker is maintained by the project development team. Thus, based on this information we remove from our study the 3,801 projects (out of the 100,000) for which the issue trackers were disabled.

We further investigate the list of projects, which include popular projects such as the Linux kernel or the Ruby on Rails framework, to ensure that most of the projects are non toy projects of substantial sizes. To this end we count the total lines of source code (LOC) in each project. Over 70% of the projects contain more than 1,000 LOC. Around 35% of the projects include more than 5,000 LOC, while more than 20% contain more than 10,000 LOC. Finally, over 600 projects contain more than 1,000,000 LOC. This distribution suggests that a significant number of the projects in the dataset are of substantial sizes. To further ensure that most of the projects are not toy projects typically created by GitHub new users, we have filtered the data to projects with at least 1,000 LOC.

[5] http://git-scm.com [6] http://dwheeler.com/sloccount

C. Research questions

Research and practice of software development is performed under various assumptions about issue tracking that have not been validated through extensive empirical studies. Thus several hypotheses can be made, and accepted, that we will attempt to (in)validate in this work:

H1 – Open source projects receive large numbers, e.g., over 1000, of issue reports.

H2 – During a software project lifecyle, developers write code for which a *distinct* community of users report issues. Thus we expect less than 50% of issue reporters to be among project developers, and vice versa.

H3 – Collaborative coding environment has a positive impact on issue reporting.

H4 – The more people are interested in a project and submit issue reports, the quicker project developers handle them.

Given the extent of our dataset, we investigate in this work the following research questions to explore and assess the relationship between various aspects of software projects and the activities in their issue trackers:

RQ1. *What is the proportion of projects that receive issue reports and how can projects be differentiated in that respect?* For this research question, we study the overall adoption of issue trackers and investigate the characteristics of projects that receive issue reports.

RQ2. *How many issues are tracked in projects whose trackers are used?* With this question, we further investigate the popularity of issue trackers by quantifying the number of issues reported in the projects.

RQ3. *What is the number of occurences of category tags in issue reports? What are the most frequently appearing categories?* In this question, we investigate the kinds of issues that are reported, and we discuss the prevalence of the different categories.

RQ4. *Who enter issues into issue tracking systems? How many of them are project team members?* We study the user-community that submits issue reports and investigate the proportion of reporters that contribute to the code base.

RQ5. *What is the relationship between utilization of issue tracking system and project success (i.e., number of forks and watchers)?* We would like to establish and assess the correlation between the success of a project and the number of issue reports that it receives.

RQ6. *Does the size of user communities impact the time-to-fix rates of bugs?* In this research question, we investigate on a larger dataset of projects a result of previous studies based on a unique development project, namely Firefox.

D. Statistical measurements

Our empirical study is based on a sample of projects. Though, to the best of our knowledge, no related study involving issue trackers has ever exploited that many projects, there is a need to ensure that, statistically, our findings are significant. To this end, we resort to common metrics in statistical analysis for assessing the statistical significance of our figures, and to confirm the existence of a correlation among data from compared artefacts.

a) The Mann-Whitney-Wilcoxon (MWW) test: The MWW test is a non-parametric statistical hypothesis test that assesses the statistical significance of the difference between the distributions in two datasets [18]. We adopt this test as it does not assume any specific distribution, a suitable property for our experimental setting.

Once the Mann-Whitney U value is computed it is used to determine the p-value. Given a significance level $\alpha = 0.001$, if p-value $< \alpha$, then the test rejects the null hypothesis, implying that the two datasets have different distributions at the significance level of α=0.001: there is one chance in a thousand that this is due to a coincidence.

b) Spearman's rho: Spearman's rho (ρ), also known as Spearman's rank correlation coefficient, is used in statistics as a non-parametric measure of statistical dependence between two variables X and Y. This measure is used without any assumption that the data is normally distributed, making it a good fit for the datasets that we investigate in this study. The values of ρ are limited to the interval $[-1; 1]$, and a perfect Spearman correlation of -1 or $+1$ occurs when each variable is a perfect monotone function of the other. The closer to 0 ρ is, the more independent the variables are.

IV. Empirical Evaluation

In this section, we report the results of our empirical study. These results are provided as responses to the research questions that were formulated in Section III-C.

A. RQ1: Adoption of Issue Trackers

In the first research question we investigate the distributions of issue trackers across projects. By default, a developer creating a project in GitHub, is provided with a source control repository and a corresponding issue tracker for the project. Nonetheless, the developer can disable the issue tracking making it inaccessible to users. Issue trackers are disabled in only 3.8% of the projects. About 30% of the projects have issue trackers containing some issues, while 66% of the projects have completely unused issue trackers. In the rest of our study, we dismiss projects with issue trackers disabled from our dataset in order to emphasize on the differences between projects that have no issue reports (albeit their issue trackers are enabled) and projects having issues.

Thus, 2 projects out of 3 do not have issue reports though their issue trackers are enabled. We thus investigate why. To this end, we correlate the presence/absence of issues, in enabled issue trackers, with different properties of a project, including its age, the size of its code base, and the number of people in the development team. We also investigate the influence of the project's owner (in general the leader) over the presence of issues.

a) Project age: We measure the age of a project solely based on its date of creation on GitHub. For each project, we count the number of weeks that have passed since it has been hosted in GitHub until June 2012. Our goal is to investigate the relationship between the duration of a project's exposure in GitHub and the presence/absence of issue reports. Figure 1 shows the distribution of project age for both datasets drawn as boxplots. The boxplots are drawn using the R statistical analysis tool. Each boxplot contains 5 main horizontal lines. From top to bottom, the first line indicates the MAXIMUM, i.e., the greatest value, excluding outliers (determined by the tool). All data points above this line are outliers. The second line indicates the UPPER QUARTILE, i.e., 25% of data points are above this line. The third line is the MEDIAN, the middle of the dataset. The fourth line is the LOWER QUARTILE, i.e, 25 % of data points are below this line. Finally, the fifth line indicates the MINIMUM, i.e., the least value, excluding outliers. Data points below this line are outliers (determined by the tool).

We find that, on average, projects without issues had been in GitHub for 70.05 weeks, with a median value of 49.57 weeks, while, on average, projects with issues had been in GitHub for 81.57 weeks with a median value of 70.04 weeks. Using the Mann-Whitney-Wilcoxon test, we have established that the difference between the two datasets is statistically significant at 0.001 significance level.

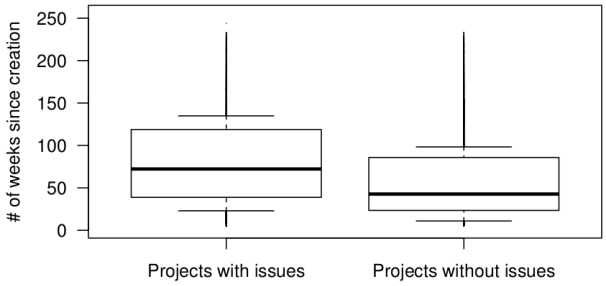

Fig. 1: Issues and the Age of Projects

This experiment reveals that projects with issues have been hosted in GitHub, in median over half a year longer than projects that do not have issue reports.

b) Lines of code: Figure 2 shows the distribution of LOCs in project code bases. Projects with issues have a median value of 1,820 LOC, while projects without issues have a median value of 1,027 LOC. We have tested and confirmed that the two distributions are significantly different using the MWW test at 0.001 significance level.

Projects with smaller code bases are less likely to be the target of issue reports. This finding is in line with the general expectation.

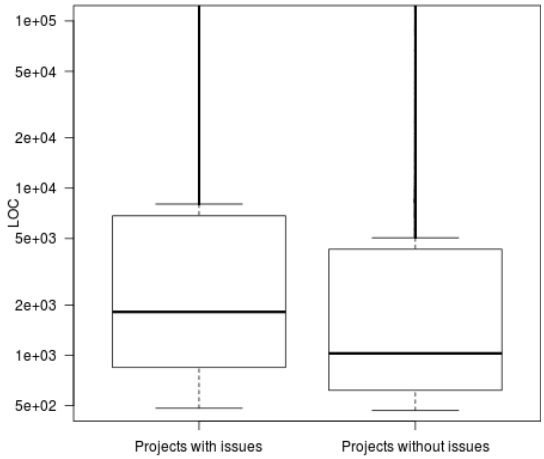

Fig. 2: Issues and Lines of Code

c) Size of development teams: We furthermore investigate the size of project teams. Figure 3 shows the distribution of the number of contributors in projects with issue reports and in projects without issue reports. The median number of developers is 2 for projects without issues (mean=43) while the number is raised to 5 for projects with issues (mean=49). We again run the MWW test and find that the difference is statistically significant at a significance level of 0.001.

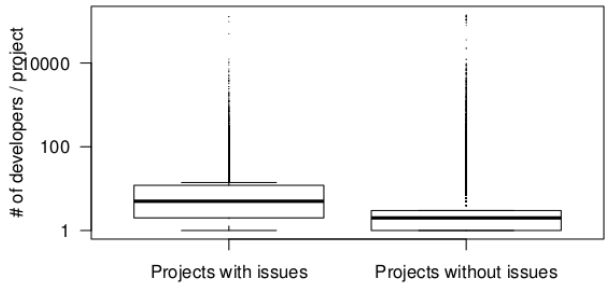

Fig. 3: Issues and Project Team Size

The conclusion in this experiment is that single-person and small-size-team projects are less likely to receive user feedback.

d) Popularity of project leader: Finally, we investigate the relationship between the popularity of project leader and the number of reported issues. We make the assumption that the project *owner*, i.e., the developer who created the project repository in GitHub, is the lead developer and that his/her popularity can be inferred from the number of developers that subscribe to his activities by *following* him. We plot in Figure 4 the distribution of the number of followers for both the owners of projects with issue reports and the owners of projects without issue reports. The median number of followers is 2 for the owners of projects without issues (mean=21), while this number is raised up to 15 for the owners of projects with issues (mean=130). We have done the MWW test and have found that the difference is statistically significant at the significance level of 0.001.

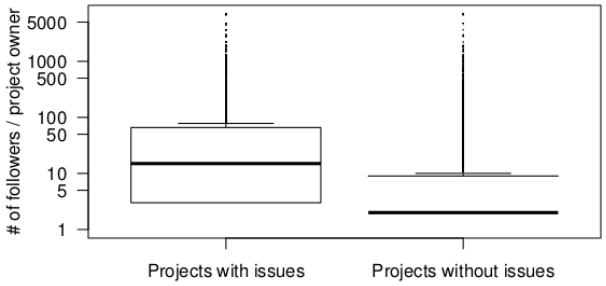

Fig. 4: Issues and Popularity of Project Owners

We find that projects owned by developers having few followers receive less user feedback than those lead by developers having more followers.

Although there may be correlations among the various dimensions that we have used in this section to investigate the presence/absence of issues in the projects, it leads to the same conclusion: smaller projects developed by small teams consisting of unknown developers do not receive much user feedback, at least not in the form of issue reports. Note that there are many such projects in the open source community.

Many projects in GitHub do not receive any issue in their issue tracking systems although these systems are automatically set up for them. This finding invalidates H1, our first hypothesis from Section III-C. *Projects with reported issues tend to be older, have more lines of code, have more number of developers in them, and have more popular owners. Project owners should be better "advertised" to other users of the social coding site GitHub which might help to cause the other users to report issues in the tracking system.*

B. RQ2: Number of Issues Tracked

In total, for projects with issue reports (20,041 projects), we have collected 803,840 reports for our analysis. We investigate in the second research question the distribution of the numbers of issues reported per project. Table I details this distribution: 86% of the projects have less than 50 issues, while only 1.14% of the projects have more than 500 issues. Almost 7% of the projects have between 50 and 100 reports while only 0.01% of the projects (i.e., 2 projects) have at least 10,000 issue reports.

Also, we investigate the numbers of issues per 1,000 LOC. For each project, we divide the number of issues with the number of kLOC. Figure 5 shows the distribution of the numbers of issues per 1,000 lines of code. The median number of issues per kLOC is 6.32. Nevertheless, the open source software projects in our dataset deal with a variety of applications domains with different complexity which can account for different bug rates. The projects also use different programming languages with disparate verbosity and a wide range of technical difficulties. We therefore investigate whether we can actually establish a correlation between the numbers of LOC and the numbers of issues.

TABLE I: Prevalence of Issues in Trackers

# Issues	# Projects	% of Projects with Issues
$0-9$	11,602	57.89%
$10-49$	5,526	27.57%
$50-99$	1,411	7.04%
$100-249$	976	4.87%
$250-499$	290	1.44%
$500-999$	163	0.81%
$1000-4999$	69	0.34%
$5000-9999$	2	0.01%
≥ 10000	2	0.01%

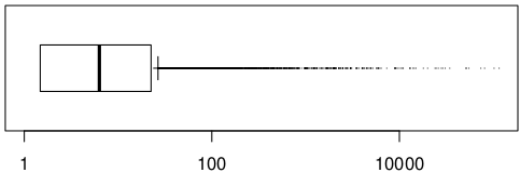

Fig. 5: Number of Issues per 1,000 LOC

In Figure 6 we provide the scatter plot of the numbers of issues and the numbers of LOC from the projects in our dataset. We compute Spearman's rho which yielded a value of 0.341. According to Hopkins [12], a Spearman's rho of between 0.3 and 0.4 indicates a moderate correlation.

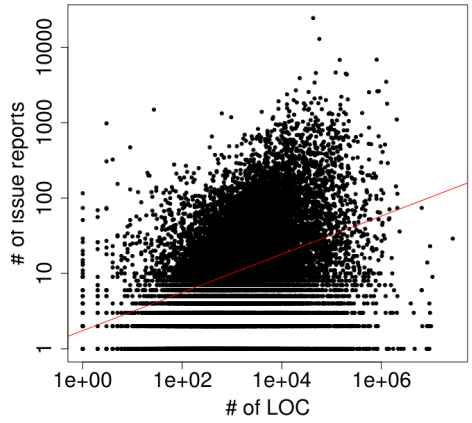

Fig. 6: Scatterplot of # of LOC and # of Issues

Most projects have only a small number of reported issues. Less than 8% of the projects have more than 100 issues in their issue tracking systems. The amount of LOC is moderately correlated with the number of issues reported.

C. RQ3: Tagging in Issue Trackers

We discuss, in the third research question, the categorization of issue reports. GitHub issue trackers provide a flexible feature for tagging issues to categorize them in order to ease their management by developers. This flexibility, however, comes with a cost since issue reporters may categorize the issues with typographical mistakes, or using various idiosyncrasies. Among the 803,840 issue reports in our dataset, 218,467 (27.17%) contain tags. Issue reporters have used in total 6,951 distinct tags to categorize their reports. Table II presents the top-10 frequent tags.

TABLE II: Top-10 Popular Tags in GitHub Issue Reports

Tag	# Tagged Issues	% of Tagged Issues
bug	40,112	18.36%
feature	22,477	10.29%
enhancement	11,584	5.30%
Win7	9,736	4.38%
ie8	7,626	3.49%
chrome	6,817	3.12%
other	6,667	3.05%
FireFox	5,669	2.59%
Feature request	5,594	2.56%
wrong-or-unclear	5,464	2.50%

There are two widespread tags, namely *bug* and *feature*, which are observed in 18.36% and 10.29% of the tagged issues respectively. Issue trackers are then essentially used in software projects to report bugs and request new features. Unfortunately, since tags are not predetermined by GitHub, many tags can be used to refer bug report or feature request. Table III lists sample tags that are used to refer to a bug report or a feature request. The variety of terms mainly stems from the preferences of issue reporters and typographical mistakes.

TABLE III: Bug Report and Feature Request Tags

bug	bug; defect; type:bug; Browser Bug; bugfix; etc.
feature	feature; request; proposal; featreq; feautre; etc

In order to more accurately quantify the number of issues belonging to the top-2 categories, we cluster the tags that are relevant to bug reports and feature requests. We first manually create two small clusters, of sizes 5 and 7, that correspond to bug reports and feature requests respectively. Next, we want to semi-automatically add more tags to these clusters. Rather than manually checking all of the thousands of tags, we want to partially automate the process. To this end, we compute the Levenshtein distance [17] between each tag and the sets of tags in our predefined clusters. Let T_t be a tag that we evaluate to include in one of our clusters, and T_c a tag that already belongs in the cluster and to which T_t is compared. We compute the Levenshtein distance d between T_t and T_c. Let the string length of T_c be l. If $d < \frac{l}{2}$, then we add the tag T_t to the cluster. Using this formula, we were able to include more tags into each cluster. At the end of the process, we manually check each tag in the clusters. This process has enabled us to identify 91 tags that belong to the category of feature requests and 18 tags in the category of bug reports. Table IV details the results after clustering tags that refer to bug/error reports and feature/enhancement requests. The updated findings suggest that bug/error reports and feature/enhancement requests are equally important for issue reporters.

The second observation from Table II is that there is a significant number of issues that are tagged with internet

TABLE IV: Bug Report and Feature Request Tags

# Label	# Labeled Issues	% of Labeled Issues
bug/error	45,123	20.65%
feature/enhancement	46,402	21.23%

browsers' names. This suggests that projects related to web development receive more issue reports than the others. This in turn may imply either one of two distinct phenomena : (1) web applications have more implementation issues than other types of software; (2) users of web applications more readily report issues than users of other types of software programs.

To assess the reality of the first phenomenon, we assume that a software application domain can be inferred from the programming language used in its code base. Though this assumption cannot be generalized, we can, to some extent, do this for web programming as many languages targeted at web development are specialized for web programming. We then attempt to establish whether projects written in common languages for web programming, such as Ruby, Python, JavaScript or PHP, contain more issues than the rest. We use David A. Wheeler's SLOCCount [29] utility to count actual physical source lines of code and their associated language. We classify the languages according to the number of issues that are filed in all the projects that are written in each language. Figure 7 shows the distribution of the number of issues for the top-10 languages that correspond to the highest number of issues. We observe that PHP is hidden in the middle with a median value comparable to that of system-level languages such as C and C++. Ruby has a slightly higher median value, while JavaScript does not appear in the top-ranked languages.

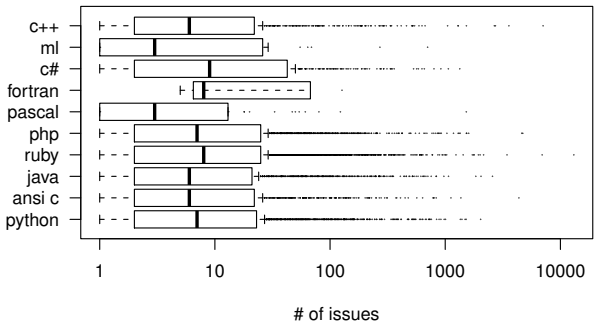

Fig. 7: Prevalence of Issue Reports for Common Languages

Thus, we dismiss the first phenomenon. We conclude that users of web applications more readily report issues. We hypothesize that since users of web applications are already accessing the web when they encounter the issues, they can directly report them to issue trackers which are online.

Only less than 30% of issue reports in our dataset are tagged. The two most common tags are bug and feature.
More than 12% of issues are tagged with labels related to web browsers and/or web applications

D. RQ4: Who Enter Issues into Issue Trackers

We now discuss the identity of issue reporters. The goal is to investigate the actual feedback provided by the user community as compared to the issues reported by development team members. Thus, for each project with issues, we analyze the number of developers, the number of issue reporters and compute the proportion of the project developers that also file issue reports and vice versa.

Based on developer names, we have identified in total 581,856 distinct developers who have contributed to the code bases of the projects with issues. Those issues were filed by 239,629 reporters. We have furthermore found that, for each project, one third of the developers (33%, the median value) have written some issue reports for the project. On the other hand, 42% of the issue reporters for a given project do not contribute to the code base.

Figure 8 shows the distribution of developers and reporters for all projects. The box plot shows that most developers in a project do not report issues in their own project. However, it indicates that a large portion of issue reporters actually contribute to the code base of the project.

Fig. 8: Proportions of Developers among Issue Reporters and Vice Versa

Project developers report issues, and issue reporters also often contribute to the code base. Our second hypothesis H2 is therefore disputed. *Development teams should therefore acknowledge reporters to encourage them to become active contributors.*

E. RQ5: Issue Trackers and Project Success

To provide insights on the fifth research question, we investigate the relationship between the success of a project and the number of its reported issues. To this end, we rely on popularity metrics based on GitHub's social coding features—*forking* and *watching*—to estimate the level of interest in a project.

a) Watchers: We first consider that the success of a project is proportional to the number of developers that *watch* the project. The score of the success metric can change as GitHub allows users to *unwatch* a project if they lose interest in it. Figure 9 shows a scatter plot that we use to explore the correlation between the number of issues and the number of watchers for each project. The Spearman's rho for the two distributions is 0.628, suggesting a large correlation between the numbers of issues and the numbers of watchers.

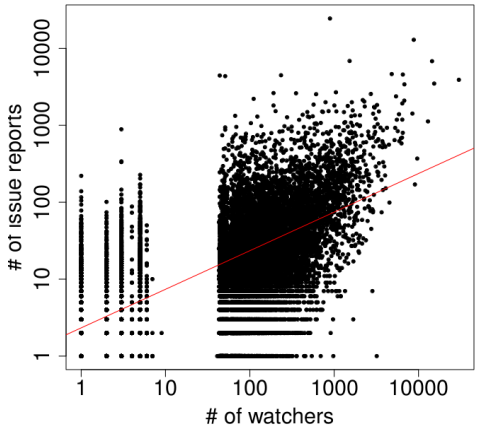

Fig. 9: Issues and Project success: Watchers vs. Issue Reports

We proceed to investigate the correlation between the numbers of watchers and the numbers of issue reporters. The scatter plot of Figure 10 shows the correlation between the two datasets. Spearman's rho amounts to 0.789. This suggests that there is a very large correlation between the numbers of issue reporters and the numbers of watchers.

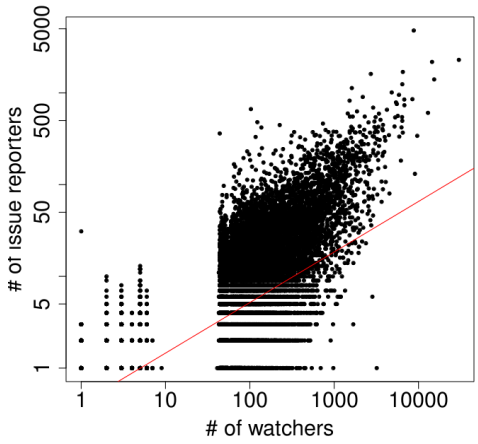

Fig. 10: Issues and Project Success: Watchers vs. Issue Reporters

b) Forks: Here we measure project success by the number of forks. Indeed, the number of forks provides a relatively good indication of the involvement of non-team developers in the development of a project.

Figure 11 shows the scatter plot of the numbers of issue reporters and the numbers of forks from the projects in our dataset. We have computed Spearman's rho which yielded the value of 0.829, suggesting a very large correlation between the number of forks and the number of issue reporters. We also compute Spearman's rho to assess the strength of the dependence between the number of forks and the number of issues as in the previous experiment. In this case, the coefficient drops to 0.669, suggesting simply a large dependence.

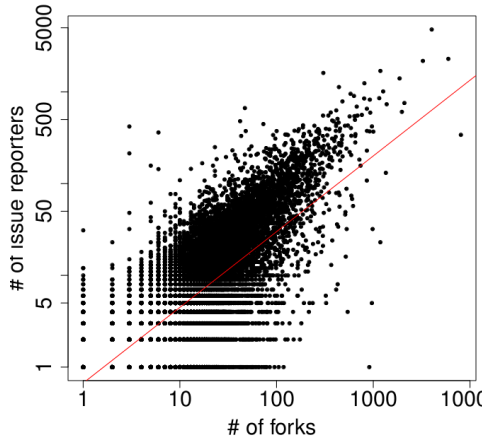

Fig. 11: Issues and Project Success: Forks vs. Issue Reporters

There is a large correlation between the number of issues and the number of watchers and forks. There is also a very large correlation between the number of issue reporters and the number of watchers and forks. Thus, distributed programming positively impacts the amount of feedback (i.e., issue reports) for a project, suggesting that this software development style should be promoted more. Our hypothesis H3 is thus validated.

F. RQ6: Does the number of reporters impact the time-to-close?

Finally, we investigate the relationship between the number of reporters and the effort made by developers to quickly address issues. In previous work based on issue reports, Van Liere had shown that the large number of Firefox issue reporters has lead to the reduction of the time-to-close interval for software defects [27]. We therefore investigate whether this is a phenomenon common among thousands of projects or whether it is specific to the development setting of Firefox.

Figure 12 shows a scatter plot of projects with their respective numbers of issue reporters and the median time-to-close intervals. A time-to-close interval is computed as the number of days between the creation date of the issue report and its close date. The graph does not show any linear dependence between those two aspects of project development. We furthermore compute Spearman's rho which yields a rho value of 0.161 confirming that there is only a small correlation among the numbers of issue reporters and the speed in which the issues are addressed.

The time-to-fix intervals of issues is only slightly impacted by the number of issue reporters. We therefore note that Hypothesis H4 is disputable.

V. Threats to Validity

We have identified the following threats to validity to our study.

a) External validity: Our *sample set of projects*, though sizeable with tens of thousands projects, may not represent the universe of all real-world projects. Furthermore, we have

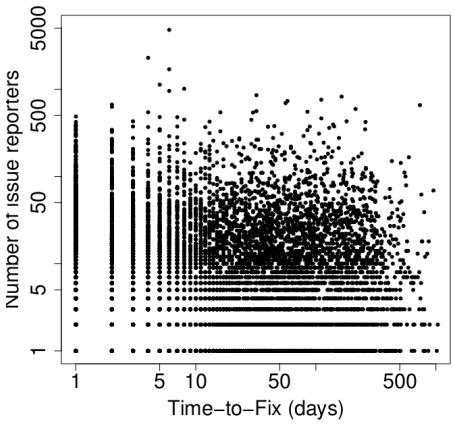

Fig. 12: Scatterplot of the Number of Issue Reporters and the Time-to-Close

focused this study on open source projects found on GitHub which may not perfectly generalize to every software project. Nonetheless, to the best of our knowledge, GitHub is the largest database of projects and has no restriction on the types of project that can be hosted. Moreover, as the study of Grammel *et al.* suggests, issue reporting in closed source projects yields similar data to that in open source projects [9].

b) Internal validity: Second, although we have *curated our datasets* to remove projects whose development, in particular issue management, occurs outside GitHub, some project owners may not disable the issue trackers even though they only deal with issues reported from other channels. This can lead to bias in our survey. Nonetheless, we have manually checked that for popularly known projects, such as Linux, that maintain issue trackers in well-known locations outside GitHub, have disabled their issue trackers on GitHub. Finally, we believe that, with a sizeable sample of 100,000 projects, the number of outliers is very small compared to the rest.

In addition, we have filtered our dataset to only consider projects with more than 10 kLOC. A different threshold may yield different findings.

c) Construct validity: Finally, we have relied on *heuristics* for estimating the popularity of a project leader, the success of a project, and the application domain of a project. As we have described for each, those are simple proxies to the ground truth. However, they provide useful metrics and good indications on the popularity of each project, the popularity of each project owner, and the domain of a project in GitHub.

VI. Related Work

GitHub, as a social coding site that hosts millions of software projects, contains a wealth of information about the practice of software development. In previous works, data from GitHub was leveraged to conduct large-scale studies on the popularity of programming languages [6] and the adoption of software testing [20]. In this paper, we exploit this data to investigate issue reporting in open source projects. To stress the importance of issue reporting we highlight in the following a number of studies related to issue reports in software development projects.

A. Acknowledging user communities

A significant number of studies have discussed the importance of the feedback provided by user communities in the life cycle of a software development project. Bagozzi *et al.* [3], Iivari [14], Hendry [10] and Singh *et al.* [22] have investigated the role of users in open source communities. In these studies they make the point that users actually drive the software project and its evolution. Furthermore, they discuss how users influence the project towards their needs and how developers acknowledge users' input.

Unfortunately, while the importance of user communities is widely accepted, the participation of users in issue tracking is more controversial in the literature. For example, considering the case of the Firefox web browser, Van Liere has concluded that large number of bug reporters reduces the time-to-fix interval for software defects [27], while Ko and Chalina reported for the same system that users file "non-issues that devolved into technical support, redundant reports with little new information, or narrow, expert feature requests" [15]. Based on our findings on thousands of open source projects, it appears that there is only a small correlation between the number of issue reporters and the time-to-fix rates of bugs.

Recently, Grammel *et al.* have explored the user involvement in issue tracking with a comparative study between open source and commercial development [9]. Basing their study on the open source Eclipse project and the closed source project IBM Jazz, they show that closed source projects can also successfully receive user feedback through issue reports as with any open source project. This suggests that our own study, which is based on open source projects, could be generalizable to closed source projects.

B. Exploring issues in software code

A large body of the literature has discussed the correlation between various properties of software code and software development processes with the presence of defects. Nagappan and Ball have shown how churns in code changes correlate with increases in software failures [19]. Koponen and Tintula, on the other hand, have investigated on Mozilla and Apache whether the changes made in project code bases were induced by defect reports. They found that this was clearly the case for Mozilla but not for Apache [16]. Posnet *et al.* have explored how new features, on the one hand, and improvement tasks, on the other hand, affect the quality of code [21].

C. Improving issue reporting

Issue reporters may file reports that are incomplete or even invalid, adding to the challenges that developers face in software projects. A number of studies have discussed and proposed approaches to enhance issue reports to make them more useful [23]. Hooimeijer and Weimer have proposed a descriptive model for measuring the quality of a bug report [11]. To build the model, they assume that the "time until resolved" is a good indicator for the quality of a bug report. Bettenburg *et al.* have later relied on actual developer feedback to train CUEZILLA [4], their utility for providing bug reporters with tips to improve the quality of their reports.

D. Enhancing issue management

Despite the various guidelines on effective bug reporting that float around the internet, development teams are flooded with a considerable number of reports with disparate quality and usefulness. Researchers have thus been devising approaches for automating various processes in the management of issues. Canfora *et al.* [7] and Anvik *et al.* [2] have proposed different approaches for automatically assigning developers to bug reports. Weiss *et al.* have proposed to automatically predict the effort for fixing bug reports and estimate the time it will take to fix them [28]. Hosseini *et al.*, in their approach for improving bug assignment, rely on the stated characteristics of a bug, such as severity, platform or priority, to predict its time-to-fix [13].

VII. Conclusion and Future Work

Issue reports are important artefacts in software development. Various issue tracking systems have been developed and widely used, among which, Bugzilla and Jira are well-known. Hosting platforms such as Freecode, Google Code, and GitHub implement in-house issue trackers to collect user feedback. The assumed prevalence of issue reports in software development projects has led to the development of approaches for various purposes, such as improving issue reporting, enhancing issue management, and leveraging issue information for bug fixing.

In this paper, we have investigated the actual adoption of issue trackers in software projects to shed lights to many research questions on the involvement of users and developers in reporting issues. We describe the findings of our empirical study on tens of thousands of open source projects in GitHub. We have found the following results:

- issues are almost exclusively reported in large projects with big development teams led by developers having the most followers;
  [H1 disputed]
- a large proportion of issue reporters are actually involved in the development of the project;
  [H2 disputed]
- distributed development, recognized with project "forking", contributes to an increase in the number of issue reporters;
  [H3 holds]
- there is only a small correlation between the numbers of issue reporters and the time-to-close delays.
  [H4 disputed]
- there is a moderate correlation between the numbers of issues tracked and the numbers of lines of code;
- social coding, through the intuitive implementation of project "watching", have an influence on the prevalence of issue reports;
- issue reporters are equally interested in reporting bugs as well as requesting new features in a project;

In this paper, we have limited our study to a *snapshot* of the projects at a given date: June 2012. In future work, we would like to investigate other questions involving *temporal* information, such as "When do people start reporting issues?", and "Do issue reports contribute to attract developers in joining the development teams?".

References

[1] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *CASCON*, 2008.

[2] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *ICSE*, 2006, pp. 361–370.

[3] R. P. Bagozzi and U. M. Dholakia, "Open source software user communities: A study of participation in linux user groups," *Manage. Sci.*, vol. 52, no. 7, pp. 1099–1115, Jul. 2006.

[4] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *FSE*, 2008, pp. 308–318.

[5] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein, "Linkster: enabling efficient manual inspection and annotation of mined data," in *FSE*, 2010.

[6] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère, "Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects," in *COMPSAC*, Kyoto, Japan, 2013, [To appear].

[7] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *SAC*, 2006, pp. 1767–1772.

[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[9] L. Grammel, H. Schackmann, A. Schröter, C. Treude, and M.-A. Storey, "Attracting the community's many eyes: an exploration of user involvement in issue tracking," in *HAoSe*, 2010, pp. 3:1–3:6.

[10] D. G. Hendry, "Public participation in proprietary software development through user roles and discourse," *Int. J. Hum.-Comput. Stud.*, vol. 66, no. 7, pp. 545–557, Jul. 2008.

[11] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *ASE*, 2007, pp. 34–43.

[12] W. G. Hopkins, *A New View of Statistics*. Sport Science, 2004.

[13] H. Hosseini, R. Nguyen, and M. W. Godfrey, "A market-based bug allocation mechanism using predictive bug lifetimes," in *CSMR*, 2012, pp. 149–158.

[14] N. Iivari, "Empowering the users? a critical textual analysis of the role of users in open source software development," *AI Soc.*, vol. 23, no. 4, pp. 511–528, Oct. 2008.

[15] A. J. Ko and P. K. Chilana, "How power users help and hinder open bug reporting," in *CHI*, 2010, pp. 1665–1674.

[16] T. Koponen and H. Lintula, "Are the Changes Induced by the Defect Reports in the Open Source Software Maintenance?" in *SERP*, 2006, pp. 429–435.

[17] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[18] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[19] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *ESEM*, 2007, pp. 364–373.

[20] K. Pavneet Singh, T. F. Bissyandé, D. Lo, and L. Jiang, "Adoption of Software Testing in Open Source Projects - A Preliminary Study on 50,000 Projects," in *CSMR*, Genoa, Italy, 2013.

[21] D. Posnett, A. Hindle, and P. T. Devanbu, "Got issues? do new features and code improvements affect defects?" in *WCRE*, 2011, pp. 211–215.

[22] V. Singh, M. Twidale, and D. Nichols, "Users of open source software - how do they get help?" in *HICSS*, jan. 2009, pp. 1 –10.

[23] J. Spolsky, *Joel on Software*. APress, 2004.

[24] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *ASE*, 2011, pp. 253–262.

[25] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *ICSE (1)*, 2010.

[26] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *CSMR*, 2012, pp. 385–390.

[27] D. W. van Liere, "How shallow is a bug? why open source communities shorten the repair time of software defects." in *ICIS*, 2009, p. 195.

[28] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *MSR*, 2007.

[29] D. A. Wheeler, "SLOCCount: Counting physical Source Lines of Code," http://www.dwheeler.com/sloccount/.

[30] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: recovering links between bugs and changes," in *FSE*, 2011.