Categorizing Bugs with Social Networks: A Case Study on Four Open Source Software Communities

Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer
Chair of Systems Design – http://www.sg.ethz.ch/ – ETH Zurich, Switzerland

***Abstract*—Efficient bug triaging procedures are an important precondition for successful collaborative software engineering projects. Triaging bugs can become a laborious task particularly in open source software (OSS) projects with a large base of comparably inexperienced part-time contributors. In this paper, we propose an efficient and practical method to identify *valid* bug reports which a) refer to an actual software bug, b) are not duplicates and c) contain enough information to be processed right away. Our classification is based on nine measures to quantify the social embeddedness of bug reporters in the collaboration network. We demonstrate its applicability in a case study, using a comprehensive data set of more than $700,000$ bug reports obtained from the BUGZILLA installation of four major OSS communities, for a period of more than ten years. For those projects that exhibit the lowest fraction of valid bug reports, we find that the bug reporters' position in the collaboration network is a strong indicator for the quality of bug reports. Based on this finding, we develop an automated classification scheme that can easily be integrated into bug tracking platforms and analyze its performance in the considered OSS communities. A support vector machine (SVM) to identify *valid* bug reports based on the nine measures yields a precision of up to 90.3% with an associated recall of 38.9%. With this, we significantly improve the results obtained in previous case studies for an automated early identification of bugs that are eventually fixed. Furthermore, our study highlights the potential of using quantitative measures of social organization in collaborative software engineering. It also opens a broad perspective for the integration of social awareness in the design of support infrastructures.**

I. INTRODUCTION

Triaging and processing bug reports is an important task in collaborative software engineering which can crucially affect product quality, project reputation, user motivation and thus the long-term success of a project. Practical experience from large open source software (OSS) projects confirms that – particularly in projects with large numbers of comparably inexperienced part-time contributors– the process of triaging, categorizing and prioritizing bug reports can become a laborious and difficult task that consumes considerable resources. Both the importance and complexity of this problem can be illustrated by a simple example: Out of the more than $64,000$ bug reports that have been resolved by the community of the MOZILLA FIREFOX project, more than $50,000$ (or $\approx 78\%$) of these reports have eventually been identified either as *duplicates* of known bugs, *invalid* reports that refer to a user error rather than a software issue or *incomplete* reports which lack basic information required to reproduce the alleged bug. The magnitude of this problem in large-scale projects calls for (semi-)automated techniques that assist bug handling communities in the triaging and prioritization of bug reports. The provision of methods which are able to automatically identify *valid* bug reports with high precision can have huge implications for practitioners of distributed software engineering: Being able to filter, assign and prioritize those bug reports that likely result in a bug fix can significantly improve the responsiveness of support communities. Furthermore, a temporary deferral of those bug reports that are likely to be duplicates, invalid or incomplete to a moderation queue can considerably alleviate the effort required for bug triaging. It can also be used to automatically enforce the adherence to community guidelines, e.g. by automatically asking original reporters to reconfirm that reported bugs are neither duplicates nor incomplete.

Due to the importance for practical software engineering, a number of different approaches for the automated classification of bug reports have been studied, among them approaches based on the automated assessment of information provided by bug reports [1–4], natural language processing [5–7], the temporal dynamics of bug handling processes [8], coordination patterns [9], or the reputation of bug reporters [10–12]. Based on a unique data set containing the full history of more than $700,000$ bug reports in four major OSS communities, in this paper we consider to what extent automated bug classification techniques can be based on *quantitative measures for the social embeddedness of bug reporters in the project's community*. We particularly address this question from the perspective of complex, evolving collaboration networks and the computation of node-centric metrics that capture structural properties like centrality and clustering.

Our contributions to the current state of research are:

- We study relations between the centrality of bug reporters and the eventual outcome of the bug triaging process. For the four OSS communities studied in this paper, we find strong evidence for the hypothesis that the centrality of users in the collaboration network is indicative for the quality of bug reports.
- We show that quantitative measures for the bug reporter's position in the collaboration network can be used for an automated classification of valid bug reports. For the four studied OSS communities, we find that this classification achieves a precision of up to 90.3% with an associated recall of 38.9%.

With this, we extend previous works that have studied automated classification of bugs that are eventually fixed. In particular, we use a more comprehensive data set, more

978-1-4673-3076-3/13/$31.00 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Software Engineering in Practice
RIGHTSLINK

sophisticated quantitative measures for user's position in the evolving structures of a community as well as a predictive modeling approach that is based on a support vector machine. In the following section, we provide a more detailed review of existing literature on automated bug classification and prediction mechanism as well social aspects of collaborative software engineering. From this we then extract a set of open research questions that are addressed in the remaining sections of this paper.

II. Social Aspects in Bug Report Processing

The distribution of contributions, the structure and evolution of collaboration networks in OSS projects, as well as their relation with individual and collective performance have been studied in a number of works. A quantitative study of the development efforts in the projects APACHE and MOZILLA has been presented in [13]. Among other aspects, the distribution of contributions across community members has been analyzed. For the APACHE project, the authors particularly validate that - while coding efforts are mainly concentrated on a small set of core developers - the bug handling and reporting process is based on a much larger community of part-time contributors.

Apart from the mere distribution of contributions, the topology of communication and collaborations between contributors is an interesting field of study. The relation between the network position of developers in bug handling communities and their success rate (in terms of the number of bugs the developers fix) has been studied in [14]. There, the authors find that developers with higher node degree fix bugs at a higher rate. Furthermore the authors provide implications for future research, calling for subsequent studies of the relation between communication structures and individual as well as team-based performance. Our work complements the study of [14] in the sense that we investigate the relation between the centrality of bug reporters and their individual performance, i.e. whether the reports are eventually found to refer to actual software issues. Our methods are based on earlier work quantifying the dynamics of social organization in OSS communities [15]. Social mechanisms underlying the impact of communication topologies on bug handling performance have been studied in [16]. There, the authors conclude that the most difficult task of successfully handling bugs is the mediation between the users and the developers of a project. Similar results have been presented by the authors of [17], whose analysis is based on the bug handling communities of two major OSS projects. Their analysis verifies that the collaborative identification of the cause of a software defect is one of the most difficult tasks that needs to be solved before bugs can be properly addressed by developers. Based on data obtained from the BUGZILLA community of the ECLIPSE project and similar to our approach, in [18] measures of communication dynamics and user centrality have been studied in networks constructed based on user comments and *CC* subscriptions. The findings suggest that the centrality of users in the communication flow networks extracted from BUGZILLA data is related to the future failure proneness of code. Similarly, the relationship between communication structures and success at the collective level has been studied in [19] and [20]. In those papers, the use of social network structures and communication deficiencies for the prediction of build failures has been proposed. Furthermore, it was found that positive team performance is related to communication structures that facilitate information dissemination. These quantitative insights about the social dimension of software engineering highlight the importance of social indicators and provide an important foundation for our approach of using related measures from social network analysis for the classification of bug report quality.

Due to the difficulty of handling user contributed bug reports in large-scale projects with millions of users, a number of different approaches for supporting bug triaging processes based on an automatic classification of bug reports have been studied. In [2] a simple linear regression model for the quality of bug reports has been proposed based on a data set of $27,984$ bug reports from the project MOZILLA FIREFOX. The model is based both on information available at the time of submission as well as post-submission data like the number of comments or attachments added during the first hours and days. The evaluation of a model based on this data shows that there is a 5% increase of predictive power compared to a pure chance prediction. In a case study on the ECLIPSE project [4], a predictive model has been introduced that is based on the textual information in comments and the bug description. The analysis shows that the model yields a precision of 62.9% and a recall of 84.5% when predicting which bugs will be reopened after being marked as closed. Apart from simple regression models, machine learning approaches have been used for the automated classification and triaging of bug reports in a number of works [1, 3, 8, 21, 22]. In [1], the use of machine learning techniques for assisting humans in assigning bugs to developers has been proposed. In [3] a machine learning approach is used to reduce bug tossing, i.e. the simultaneous assignment of bugs to multiple developers. The authors show that bug tossing can be reduced significantly when classifying developers according to the product relationships of previously fixed bugs. In [22] different machine learning approaches have been applied to bug descriptions and comments stored in the BUGZILLA database of the ECLIPSE project. Here the authors prove the suitability of support vector machines and Latent Dirichlet Allocations for the prediction of the category of bug reports.

Indicators for the *social context* of users have been considered for the prediction of which bugs get fixed and which are likely to be reopened in [10, 12]. In [10], a number of bug report features have been used, including the reputation of bug reporters in terms of the fraction of their previously reported bugs that were eventually fixed. The authors show that a statistical model for the automated identification of those bugs that will get fixed can yield a precision of 68% and a recall of 64%. The same approach has recently shown to be successful for the prediction which bugs get reopened [12].

Data from the BUGZILLA installations of ECLIPSE and

RIGHTSLINK

MOZILLA have been used in [11] to model developer prioritization in bug repositories. Here the authors used a ranking of developers based on social networks and apply a support vector machine to predict the severity of bug reports assigned to developers. In [23], a predictive model for the bug severity based on the location of the defect in the software dependency network has been studied. Here the authors find that the degree of components in the software is indicative for bug severity.

III. STUDY DESIGN AND METHODOLOGY

Based on a the review of existing work that is related to a) the influence of social embeddedness on the performance of communities and individual contributors and b) the automated classification of bug reports, we identify the following open research questions which will be addressed in our paper:

RQ1 Is the position of bug reporters in the evolving collaboration structures of bug handling communities related to the quality of contributed bug reports?

RQ2 Can quantitative measures for the position of bug reporters be used to predict which bug reports refer to valid bugs?

With the prediction methodology proposed in section V, we extend and improve previous approaches to automated bug classification in a number of ways: First we consider a larger data set which contains a total of more than 5.8 million time-stamped change events for more than $700,000$ bug reports from four large OSS projects. Second, rather than using simple, one-dimensional social indicators like the number of previously submitted reports or the number of connections, we use a set of nine topological measures to quantify the position of bug reporters in the collaboration network, among them a comprehensive set of centrality measures as well as degree, local clustering structure and membership in the largest network component. Third, rather than taking a simple static perspective, we consider *evolving collaboration networks* by using fine-grained temporal data on collaboration and communication events. Based on these features, we apply a machine learning approach for predicting which of the bug reports are eventually identified as valid, i.e. which are referring to actual bugs that need to be addressed by the community. We further strictly limit our prediction methodology to *only include information available at the time of the submission of a bug report, thus making the approach directly applicable in a practical setting.* To the best of our knowledge, no prior work has combined such a comprehensive set of network measures on evolving networks with a machine learning classifier and applied it to data set of similar scale. Our findings show that our methods significantly improve the precision and recall of existing automated bug classification schemes.

In our paper, we adopt a data-driven approach that is based on a data set we collected from the MOZILLA BUGZILLA[24] installations of the four communities evolving around the following OSS projects: MOZILLA FIREFOX, MOZILLA THUNDERBIRD, ECLIPSE and NETBEANS. In the following, we provide a detailed description of a) the data retrieval process and the categories of bug reports available in the data, b) our methodology of extracting time-stamped collaboration networks and c) the measures applied in our analysis.

A. Data Retrieval

Records retrievable via the BUGZILLA *API* are centered around *bug reports* which are identified by a unique *bug Id*. Further, users registered in the BUGZILLA installation of the respective OSS project are also identified by their unique *user Id*. Each bug report has a number of associated fields, for which the history of all updates along with a time stamp and the *Id* of the user who has changed the field, is stored in the database. For our analysis, we use the *user Id* of the user who initially submitted the bug report (throughout the paper we will refer to this user as the *bug reporter*), the time stamp of the initial submission, and the status of the bug report (like e.g. *unconfirmed*, *pending*, *reproduced*, *resolved*). We further use the *user Id* of the so-called *ASSIGNEE*, who is a user responsible for providing a fix to the bug, and a list of *user Id's* of those users that have (or were) subscribed to receive subsequent updates on the bug report, *CC*.

For our study, we retrieved the full history of all bug reports via the *API* of the respective projects. Our data set contains roughly $715,000$ bug reports and 5.8 Million change events recorded in the time between January 1999 and June 2012. Table I presents some basic statistics of the data set used throughout this paper.

In particular, our analysis is focused on a subset of those $287,540$ bug reports that had a final status indicating that they were *resolved*. We limit our analysis to these bug reports because the bug handling community already completed the triaging process and thus reached a decision on how they were processed. For this subset of resolved bugs we extract the full history of change events and categorize each bug according to the final change in the *Resolution* field of the corresponding record. Bugs that had a final *Resolution* status of *FIXED* (i.e. a bug fix has been created by a developer), *INVALID* (i.e. the report refers to expected behavior or wrong usage rather than to a software bug), *DUPLICATE* (i.e. the report refers to a bug that has already been reported) or *WONTFIX* (i.e. the bug is valid and reproducible but it will not be fixed due to a lack of resources or low priority) were categorized accordingly. In addition, we consider a bug report to fall into the category *INCOMPLETE* whenever it had an intermediate status that indicates that the initial bug report was missing information required to properly triage the bug. While the projects MOZILLA FIREFOX, MOZILLA THUNDERBIRD and NETBEANS make use of a specific status for incomplete reports, in the ECLIPSE community, bug reports that lack necessary information simply remain in the initial status *NEW*. Since this procedure does not allow us to easily classify corresponding bugs, we disregard the *INCOMPLETE* category for the ECLIPSE project.

B. Network Construction

Our approach to utilize measures for the embeddedness of users in the community is based on the extraction of social

RIGHTSLINK

TABLE I
TIME PERIODS, NUMBER OF BUGS, NUMBER OF CHANGE EVENTS AND NUMBER OF BUGS WITH PARTICULAR STATUS. THE DIFFERENT BUG RESOLUTION CATEGORIES ARE THE FOLLOWING: *FIX* FOR FIXED, *DUP* FOR DUPLICATE, *INV* FOR INVALID, *WOF* FOR WON'T FIX AND FINALLY *INC* FOR INCOMPLETE. MORE DETAILS IN SECTION III-A.

	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS	Total
Start date	April 2002	January 2000	October 2001	January 1999	–
Total bug reports	112,968	35,388	356,415	210,921	715,692
Change events	1,068,070	313,957	2,594,385	1,875,878	5,852,290
Changes / report	9.45	8.87	7.28	8.89	8.18
Resolved bugs (resolved/total)	64,088 (0.57)	21,644 (0.61)	158,957 (0.45)	42,851 (0.19)	287,540 (0.40)
FIX (FIX / resolved)	10,856 (0.17)	4,508 (0.21)	103,453 (0.65)	21,442 (0.50)	140,259 (0.49)
DUP (DUP / resolved)	24,263 (0.38)	10,336 (0.48)	28,227 (0.18)	9,328 (0.22)	72,154 (0.25)
INV (INV /resolved)	11,785 (0.18)	2,829 (0.13)	12,601 (0.08)	4,082 (0.10)	31,297 (0.11)
WOF (WOF / resolved)	2,708 (0.04)	581 (0.03)	14,676 (0.09)	5,515 (0.13)	23,480 (0.08)
INC (INC / resolved)	14,476 (0.23)	3,390 (0.16)	-	2484 (0.06)	20,350 (0.07)

networks. Those can be viewed as proxies for the collaboration and communication structure of an OSS project during a particular period of time. Our data set is comprehensive in that it contains a history of all events associated with all bugs reported during a period of more than ten years. For the construction of social networks we focus on those update events that directly capture dyadic interactions, and therefore can arguably be interpreted as pairwise interactions between users. In particular, we use the dyadic relations *ASSIGN* and *CC* for this purpose. For the present study, we decided to neglect additionally available information like the sequence of comments on bugs for which the inference of direct interactions between users is more difficult and necessarily error-prone. Any user can add usernames to the *CC* list of a bug report, which will make sure that the added user receives information on all future updated of a particular bug. Special permissions are required for a user to *ASSIGN* a bug to another user, which is hence being made responsible for providing a solution for the issue. We would like to emphasise that focusing on *CC* and *ASSIGN* updates necessarily provides a limited perspective on the interactions between users. Nevertheless we argue that the generated social networks are insightful with respect to the collaboration and communication structures of a project: A *CC* interaction between users A and B indicates that A is aware of B and that A knows what B is interested in. In addition, an *ASSIGN* interaction between A and B is indicative for different roles in the community. For example, user A identifies the cause of a bug and assigns it to user B who is a developer and likely be able to fix it.

The simplest, and usually adopted, approach to analyze social networks in OSS communities is to study the topology by aggregating all interactions throughout the history of a project. However, since our data set covers interactions from more than one decade, the meaningfulness of such aggregated structures is questionable. It is likely that most of the users represented by nodes in the aggregated network never have been active within the same time period. This clearly limits the expressiveness of the network structure in terms of a project's "social organization". In order to overcome this shortcoming, we make use of the fact that - like all other updates in our data set - *CC* and *ASSIGN* interactions have a precise time stamp. In our analysis, we particularly study networks of collaborations constructed by aggregating all interactions occurring within time windows with a length of 30 days. This allows us to focus on collaboration networks existing at short periods of time during the project's history, e.g. when particular users were present, particular bugs were reported or when the project had a particular level of popularity and maturity. In the following, we provide a detailed description of the quantitative measures used in our analysis of the resulting time-stamped collaboration networks.

C. Network Measures

The literature is rich in measures to quantify structural features of (social) networks [25, 26]. We adopt some of these measures to capture the social organization in bug processing communities.

1) Centrality Measures: Node-centric measures of *centrality* allow us to assess the relative *importance* of nodes in a given network. This importance, or centrality, can be expressed through different approaches. The simplest one is the number of connections a node has to other nodes, known as the *degree centrality*. In a social context, degree centrality can be interpreted either in terms of the potential impact of a node on other nodes or as the amount of information available to a node. However, degree centrality does not capture the actual *position* of a node in the network in terms of how close an node is to *all* other nodes. A further important measure is thus the so-called *closeness centrality* [27], which is defined as the inverse of the sum of all distances to all other nodes. The centrality of nodes can be also measured in terms of the role they play in connecting other nodes. The so-called *betweenness centrality* is given by the total number of shortest paths between all possible pairs of nodes that pass through a node [25].

Eigenvector centrality is a more sophisticated feedback centrality measure in which the centrality of a node is recursively influenced by the centrality of its direct neighbors:

$$Ev(n_i) = \frac{1}{\lambda} \sum_{n_j \in M(n_i)} Ev(n_j)$$

Here $M(n_i)$ is the set of direct neighbors of node n_i and λ is the largest eigenvalue of the network's adjacency matrix A [26]. In other words, nodes connected to highly central nodes increase their own centrality. For our analysis, we use the eigenvector centrality implementation of the IGRAPH library

RIGHTSLINK

[28] for the R language [29]. The last two measures considered are the *clustering coefficient* and *k-coreness*. The first captures to what degree two nodes that have a neighbor in common are also neighbors. The second one is based on a network decomposition such that nodes are assigned to so-called *shells* of the network topology. Nodes belong to a given shell k if they have a degree k after removing all other nodes with degree up to $k-1$. Nodes in shells with higher number can be seen to have higher relative influence within a community [30].

2) Analysis of Largest Connected Component: In large-scale, sparse social networks usually not all nodes have a link to the rest of the network, i.e. some parts can be isolated. Thus, in addition to connected parts (components) of the network, a number of disconnected components exist. Several network measures, including eigenvector centrality, are not well defined for networks with different connected components. To overcome this problem, we restrict our analysis to the so-called *largest connected component* (LCC) of the monthly collaboration networks. We find that the fraction of nodes in the LCC was high: For ECLIPSE, an average fraction of 0.78 of all users in the monthly collaboration network belong to the LCC, for NETBEANS the average fraction is 0.96, for MOZILLA THUNDERBIRD 0.53 and for MOZILLA FIREFOX 0.58. Moreover, we verified that the largest size of the remaining components was insignificant when compared to the size of the LCC. To illustrate our approach, in Figure 1 we show the components of a monthly collaboration network for each of the four projects studied in our analysis. In each of these networks of comparable size the LCC is highlighted. Structural differences between these networks indicate significant variations in the social organization of the four projects.

IV. USER CENTRALITY AND BUG REPORT QUALITY

In this section we apply the methods introduced in section III to address research question **R**Q1, specifically:

Is the centrality of bug reporters in the collaboration network related to the quality of the submitted bug reports?

A positive answer to this question could serve as a foundation for the development of automated bug classification schemes that are based on methods from social network analysis. We investigate this question for four major OSS projects that adopt the BUGZILLA bug tracking system: ECLIPSE, NETBEANS, MOZILLA FIREFOX and MOZILLA THUNDERBIRD. Using the data set described in section III-A, we analyze the history of all bugs that were eventually marked as *resolved*, along with the corresponding resolution categories. As emphasized before, the *resolved* bugs are the ones for which the bug report processing was completed (see section III-A for details). The resolution categories are: *FIXED*, *DUPLICATE*, *INVALID*, *WONTFIX* and *INCOMPLETE*. In addition, we consider bugs to fall in the category *INCOMPLETE*, if a bug report had this status at some point in its history, independently of the final resolution category. According to the bug handling guidelines of the respective communities, bug reports will only be marked as such if the reporter failed to include the required additional information within a certain period of time. Some basic statistics about the total and relative number of bugs falling in the different categories are given in Table I.

In line with our research question, we first hypothesize that the submission of "helpful" bug reports - those that eventually result in a bug fix - increases the centrality of the bug reporter, i.e.

H1: *The centrality of users increase after the submission of bug reports that eventually result in a bug fix.*

Complementary to **H**1 we can furthermore hypothesize:

H2: *The centrality of users decrease after the submission of bug reports that are eventually identified as duplicate or invalid.*

While these two hypotheses address the relation between the submission of helpful or duplicate bug reports and *subsequent changes* of the users' centrality in the community, it is also reasonable to consider an inverse dependence: The users' centrality at the time when a bug is reported can possibly influence their ability to contribute helpful bug reports. A better knowledge of bug handling procedures that results from a higher centrality in the community may for instance help to prevent duplicate bug reports. In our third hypothesis - which is also the basis for our prediction method - we thus propose that the centrality of bug reporters is indicative for the outcome of the bug handling process.

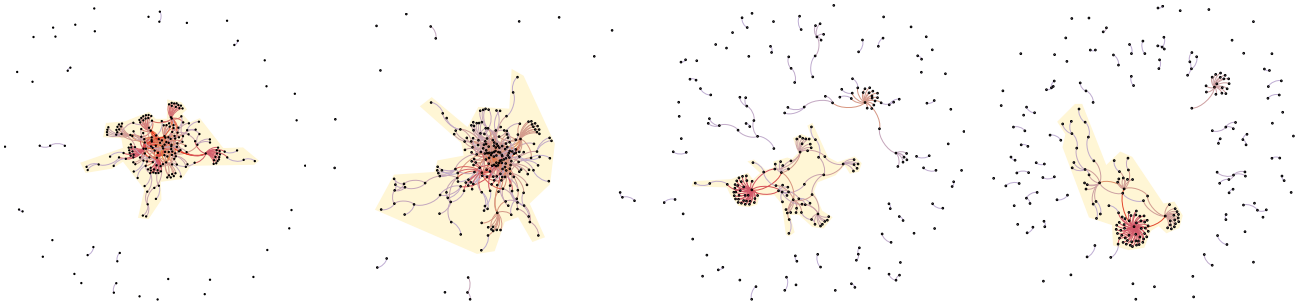
H3: *The centrality of a bug reporter in the monthly collaboration network preceding the time of the report is indicative for the eventual outcome of the bug handling process.*

We would like to emphasize that one can imagine different mechanisms, both at the level of the user and the community that are compatible with these hypotheses. As mentioned above, the users' centrality in the network is likely to be correlated with the level of contribution as well as the knowledge and experience of contributors. These factors are likely to influence the quality of bug reports submitted by a user. Furthermore, being central in the community can influence the attention received by other users, thus increasing the chance of bug reports being taken seriously, prioritized and eventually fixed.

A. Analysis

We test hypotheses **H**1, **H**2 and **H**3 in the following way: We first categorize all bug reports that were eventually resolved according to their final resolution. As described in section III-B, we then extract the collaboration networks in the month preceding and following the time of the bug report and compute the eigenvector centrality of bug reporters in both networks. By this, we obtain five distributions of centralities of bug reporters in the monthly collaboration network *preceding* the time of the bug report for the bug categories *FIXED*, *DUPLICATE*, *INVALID*, *WONTFIX* and *INCOMPLETE*. We denote these as FIX_1, DUP_1, INV_1, WOF_1 and INC_1 respectively. Similarly, we extract the distributions of eigenvector centralities of bug reporters in the month *after* the bug report and denote these as FIX_2, DUP_2, INV_2, WOF_2 and INC_2. We would like to emphasize that - out of the

RIGHTSLINK



(a) ECLIPSE (Dec. 2002) 244 nodes, 319 links
(b) NETBEANS (Jun. 2006) 246 nodes, 513 links
(c) FIREFOX (Oct. 2003) 241 nodes, 184 links
(d) THUNDERBIRD (Nov. 2004) 245 nodes, 170 links

Fig. 1. Four monthly collaboration networks representing the communities of ECLIPSE, NETBEANS, FIREFOX and THUNDERBIRD. Although the networks are of similar size, the different topological structures indicate that these communities differ largely in terms of social organization. The yellow shaded area represents the network's largest connected component (LCC).

TABLE II
COMPARISON OF EIGENVECTOR CENTRALITY DISTRIBUTIONS FOR THE FIVE BUG RESOLUTION CATEGORIES CONSIDERED IN OUR ANALYSIS. IN EACH ROW WE PRESENT THE HYPOTHESIS BEING TESTED, THE CORRESPONDING DISTRIBUTIONS INVOLVED (E.G. $FIX_1 \sim FIX_2$), THE ALTERNATIVE HYPOTHESIS (I.E. $>$,$<$,$\neq$), ITS RESPECTIVE p-VALUE (WE INDICATE WITH (*) WHEN WE ACCEPT THE ALTERNATIVE HYPOTHESIS) AND THE SAMPLE SIZE OF EACH DISTRIBUTION (I.E. NUMBER OF BUGS). MORE DETAILS IN SECTION IV-A.

Hypothesis	Comparison of Distrib.	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS
H1	$FIX_1 \sim FIX_2$	$<$, $p = 0.0026$, (*) (5847, 6140)	$>$, $p = 0.0351$, (*) (2139, 2377)	$\neq$, $p = 0.1453$ (66208, 69026)	$\neq$, $p = 0.6435$ (13930, 14668)
H2	$DUP_1 \sim DUP_2$	$>$, $p = 0.0349$, (*) (6799, 8697)	$>$, $p < 2.22e-16$, (*) (973, 3027)	$>$, $p < 2.22e-16$, (*) (17600, 22215)	$>$, $p < 2.22e-16$, (*) (3984, 5470)
H2	$INV_1 \sim INV_2$	$\neq$, $p = 0.7268$ (1321, 1394)	$>$, $p = 0.0449$, (*) (242, 297)	$\neq$, $p = 0.8489$ (5313, 5958)	$\neq$, $p = 0.1266$ (1906, 2066)
H3	$FIX_1 \sim WOF_1$	$>$, $p = 1.81e-10$, (*) (5847, 1022)	$>$, $p = 1.58e-06$, (*) (2139, 106)	$<$, $p < 2.22e-16$, (*) (66208, 7769)	$>$, $p < 2.22e-16$, (*) (13930, 2847)
H3	$FIX_1 \sim DUP_1$	$>$, $p < 2.22e-16$, (*) (5847, 6799)	$>$, $p < 2.22e-16$, (*) (2139, 973)	$<$, $p < 2.22e-16$, (*) (66208, 17600)	$>$, $p < 2.22e-16$, (*) (13930, 3984)
H3	$FIX_1 \sim INV_1$	$>$, $p < 2.22e-16$, (*) (5847, 1321)	$>$, $p = 4.93e-10$, (*) (2139, 242)	$<$, $p < 2.22e-16$, (*) (66208, 5313)	$>$, $p < 2.22e-16$, (*) (13930, 1906)
H3	$FIX_1 \sim INC_1$	$>$, $p < 2.22e-16$, (*) (5847, 587)	$>$, $p < 2.22e-16$, (*) (2139, 159)	(-)(-) (66208, 0)	$>$, $p < 2.22e-16$, (*) (13930, 661)

quantitative measures introduced in section III-C - in this section we only use eigenvector centrality to quantify the position of bug reporters. However, for the classifier proposed in the next section we use a more comprehensive set consisting of additional topological measures for centrality, coreness, degree and membership in the LCC.

In order to compare the different eigenvector centrality distributions of bug reporters described above, we apply a *Wilcoxon-Mann-Whitney* test [31]. For two samples S_A and S_B drawn from two distributions F_A and F_B with $F_A(x) = F_B(x - \alpha)$, the *Wilcoxon-Mann-Whitney* infers the stochastic ordering of the distributions, i.e. whether the shift parameter α is likely to be larger than zero (i.e. $F_A > F_B$) or smaller than zero (i.e. $F_A < F_B$). Based on the null hypothesis that $\alpha = 0$ (i.e. $F_A \sim F_B$) the test is executed either with the one-sided alternative hypotheses $F_A > F_B$ or $F_A < F_B$, or with a two-sided alternative hypothesis $F_A \neq F_B$. For each of the three alternative hypotheses, the test yields a p-value which - if it is below a given significance threshold - is used to reject the null hypothesis in favor of the alternative hypothesis. If none of the p-values for one of the alternative hypotheses is below the significance threshold, one cannot reject the *null hypothesis* that both samples S_A and S_B are in fact drawn from the same distribution, i.e. $F_A \sim F_B$.

We now test **H**1 by applying the methodology described above to the two samples FIX_1 and FIX_2, i.e. we test whether there is an increase in the eigenvector centralities of users after the report of a bug that is eventually fixed. The null hypothesis **H**0 related to **H**1 is that the samples FIX_1 and FIX_2 are drawn from the *same distribution*, i.e. $FIX_1 \sim FIX_2$ or - in other words - the eigenvector centrality of users reporting helpful bugs *does not change* after the time of the report. We reject the null hypothesis and accept hypothesis **H1** if the p-value for $FIX_1 < FIX_2$ is below a significance threshold of 0.05. The resulting p-values for the comparison of the distributions FIX_1 and FIX_2 are given in Table II. One observes that for the projects ECLIPSE and NETBEANS one cannot reject the null hypothesis that eigenvector centralities of users *do not change* after the submission of bug reports that result in a bug fix. However, for MOZILLA FIREFOX *there is a significant increase* in the eigenvector centralities of users reporting bugs that are eventually fixed. Interestingly, for MOZILLA THUNDERBIRD *we also reject the null hypothesis but instead find a significant decrease of eigenvector centrality*.

Similar to **H**1, we test hypothesis **H**2 by applying a *Wilcoxon-Mann-Whitney* test on the samples DUP_1, INV_1, DUP_2 and INV_2, i.e. we compare the eigenvector centrality distributions of bug reporters submitting duplicate or invalid bug reports *before* and *after* the time of the submission.

RIGHTSLINK

The results shown in Table II provide strong evidence for hypothesis **H**2 regarding bugs that are eventually identified as duplicates. In fact, the null hypothesis that DUP_1 and DUP_2 are drawn from the same distribution can be rejected in favor of the alternative hypothesis $DUP_1 > DUP_2$ for all of the studied projects. For the case of bugs that are eventually identified as invalid, we cannot reject the null hypothesis for the projects FIREFOX, ECLIPSE and NETBEANS. For the project THUNDERBIRD the null hypothesis can be rejected in favor of hypothesis **H**2.

Finally, we test hypothesis **H**3 by comparing the distribution FIX_1 to the distributions WOF_1, DUP_1, INV_1 and INC_1, i.e. we check whether the centralities of users reporting bugs that are eventually fixed are - on average - different than of those reporting bugs that fall in other categories. The results of our analysis are shown in Table II. We find strong evidence for hypothesis **H**3 when comparing FIX_1 to either WOF_1, DUP_1, INV_1 or INC_1. In the projects FIREFOX, THUNDERBIRD and NETBEANS we particularly find that the centrality of users reporting bugs that are eventually fixed is significantly larger. Interestingly, the opposite relation holds for the project *Eclipse*, i.e. here the centrality of users reporting bugs that are eventually fixed is significantly smaller.

In summary, our analysis validates that there is a statistically significant relation between the centrality of a bug reporter and the outcome of bug handling processes. We particularly emphasize that our analysis supports the hypothesis that the centrality in the collaboration network during the month preceding the bug report is indicative for the outcome of the bug handling process. In the following section, we make use of this finding to develop a prediction method that can e.g. be applied in (semi-)automatic bug report prioritization strategies. By this, we show that a quantitative analysis of social structures in OSS communities can assist in bug triaging. While in the next section we exclusively focus on the use of a set measures of *social embeddedness*, we would like to highlight that a combination of these measures with existing methods is likely to further improve the classification mechanism.

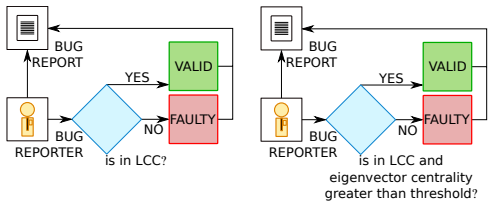
V. CLASSIFICATION OF BUGS WITH SOCIAL NETWORK ANALYSIS

Based on the observed relations between the bug reporters' centrality and bug report quality presented in section IV, we now address research question **RQ**2, specifically:

Can quantitative measures for the position of bug reporters be used to predict which bug reports refer to valid bugs?

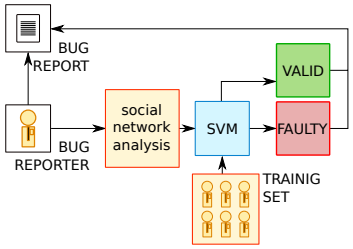
The goal is to develop a practical method that makes use of topological measures for the position of bug reporters in the collaboration network. In order to facilitate the bug triaging process, we particularly aim at predicting whether a bug report is likely to be either *Valid* or *Faulty*. As *Valid* bug reports, we consider all bug reports that have a final status of *FIXED* or *WONTFIX*. Conversely - and in line with the semantics of bug categories provided in section III-A - we consider all bug reports as *Faulty* that have a final status of *DUPLICATE*, *INVALID* or *INCOMPLETE*.

The task for our classifier is to predict whether a given bug report is *Valid* or *Faulty*, based on a set of features that are comprised of different quantitative measures for the position of bug reporters in the collaboration network. In order to highlight the predictive power gained by the inclusion of further measures, we start with a very simple classifier which only considers the presence of a bug reporter in the network's largest connected component (LCC). We then incrementally add a prediction that is based on a threshold of eigenvector centrality as well as - eventually - a support vector machine that makes use of the following set of nine topological measures calculated at the level of a node: presence in the LCC, eigenvector, betweenness, and closeness centrality, local clustering coefficient, coreness, as well as in-, out- and total degree. Illustrative overviews of the three different classification schemes are provided in Figures 2(a) - 2(c). For each of the obtained classifiers, we evaluate its predictive power in terms of *precision*, *recall* and the corresponding *F*-score (i.e. equally weighted precision and recall) [2, 32]. In order to enable the reader to correctly interpret the predictive power based on the obtained precision and recall values, in the first line of Table IV we indicate the actual fraction of *Valid* bug reports in our data set for each of the considered projects.



(a) Classifier based solely on LCC membership

(b) Thresholding on eigenvector centr. of LCC nodes



(c) SVM Classifier

Fig. 2. Graphical illustration of the three classifiers described in section V. When bug reporters submit reports, we immediately quantify the nine measures that express their social embeddedness as described in the text. These are used as input to the classifier, which will then predict if bug reports are valid or faulty. For the case of the SVM classifier, we separate 5.0% of the samples to be used as training data.

We first consider a simple prediction method which considers a bug report to be valid whenever the bug reporter is in the LCC of the collaboration network in the month preceding the submission of the bug report. The basis for this prediction is provided in Table III, which lists the fraction of bug reporters belonging to the LCC of the network individually for each of the different bug categories. In the two bottom rows, we furthermore provide the same values for the aggregated sets of *Valid* and *Faulty* bugs. For MOZILLA FIREFOX and MOZILLA

RIGHTSLINK

TABLE III
PERCENTAGES OF BUG REPORTERS THAT ARE IN THE LCC OF THE SOCIAL NETWORK IN THE MONTH PRECEDING THE REPORT. THE PERCENTAGES GIVEN WERE CALCULATED FOR EACH OF THE RESOLUTION CATEGORIES (E.G. FOR FIREFOX, FROM THOSE THAT REPORTED BUGS RESOLVED AS FIXED: 53.9% WERE IN THE LCC WHILE 46.1% WERE NOT).

	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS
FIX	53.9%	47.4%	64.0%	65.0%
DUP	28.0%	9.4%	62.4%	42.7%
INV	11.2%	8.6%	42.2%	46.7%
WOF	37.7%	18.2%	52.9%	51.6%
INC	4.1%	4.7%	-	26.6%
Valid	50.6%	44.1%	62.6%	62.2%
Faulty	17.2%	8.3%	56.1%	41.2%

THUNDERBIRD one observes a significant difference between these two categories, i.e. the fraction of reporters of *Valid* bugs that are in the LCC is significantly higher than the fraction of reporters of *Faulty* bugs. For ECLIPSE and NETBEANS the effect is less pronounced. Table IV (i.e. (LCC) rows) shows the precision, recall and F-score of a classifier that is solely based on LCC membership. When comparing to the real proportion of *VALID* bug reports, this predictor clearly performs better than a null model of randomly sampling bug reports. Due to the stronger effect of LCC membership, the performance is clearly better for MOZILLA FIREFOX and MOZILLA THUNDERBIRD, which at the same time are the projects with the smallest proportion of *VALID* bug reports.

TABLE IV
PRECISION (p), RECALL (r) AND F-SCORE OF FILTERING VALID BUG REPORTS BASED ONLY ON MEASURES OF SOCIAL EMBEDDEDNESS.

	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS
Valid	21.0%	23.3%	74.3%	62.4%
p (LCC)	44.1%	62.1%	76.3%	71.9%
r (LCC)	50.9%	44.5%	62.6%	62.4%
F (LCC)	0.47	0.52	0.69	0.67
p (evcent)	60.4%	68.6%	76.3%	76.7%
r (evcent)	30.5%	5.4%	62.6%	38.8%
F (evcent)	0.41	0.10	0.69	0.52
p (SVM)	82.5%	90.3%	88.7%	78.9%
r (SVM)	44.5%	38.9%	91.0%	87.0%
F (SVM)	0.58	0.54	0.89	0.83

As the next measure we add to the classifier the eigenvector centrality of bug reporters. This classifier will mark bug reports as *VALID* if the reporting users is part of the LCC and if their respective eigenvector centrality scores are above a precentile threshold that is tuned for each community individually. The results shown in Table IV (i.e. (evcent) rows) indicate that - compared to a classification based on mere LCC membership - the inclusion of eigenvector centrality increases the precision while generally decreasing recall and F-score. Due to the negative relation between eigenvector centrality and bug report quality found for MOZILLA THUNDERBIRD, the drop in the F-score is particularly pronounced for this project.

Our next and final step towards a practical tool is a) the use of a support vector machine (SVM) [33] for the prediction of *valid* bug reports and b) the use of the full set of nine topological measures. In order to eliminate the risk of overfitting the data, we use a training set that is composed of only 5.0% of all available samples. The nine measures we consider as input features are: *LCC membership*, *eigenvector centrality*, *betweenness centrality*, *total degree*, *in-degree*, *out-degree*, *closeness centrality*, *clustering coefficient* and *k-coreness*. We present the results of the SVM classifier in Table IV (i.e. (SVM) rows). For MOZILLA FIREFOX and MOZILLA THUNDERBIRD we obtain precision values of 82.5 and 90.3 as well as F-scores of 0.58 and 0.54 respectively. In both of these projects the fraction of *Valid* bug reports is comparably small (with 21% and 23.3% respectively).

The fraction of *Valid* bugs in the ECLIPSE and NETBEANS projects is significantly higher. We hypothesize that this is due to more stringent bug reporting procedures and a higher technical proficiency of users which is related to the fact that both projects target a user community that mainly consists of developers. For ECLIPSE and NETBEANS our classifier obtains a precision of 88.7% and 78.9% with F-scores of 0.89 and 0.83 respectively. Since the majority of bug reports in these two projects are *Valid*, we propose to use the classifier to identify the minority of *Faulty* bug reports instead. In Table V, we show the corresponding results for all four projects. In this setting, our classifier achieves F-scores of 0.92 and 0.91 and a precision of 86.9% and 84.9% for MOZILLA FIREFOX and MOZILLA THUNDERBIRD respectively. For the projects ECLIPSE and NETBEANS we obtain a precision of 73.6% and 73.1% and F-scores of 0.69 and 0.67 respectively.

TABLE V
PRECISION (p), RECALL (r) AND F-SCORE OF FILTERING FAULTY BUG REPORTS BASED ONLY ON MEASURES OF SOCIAL EMBEDDEDNESS.

	FIREFOX	THUNDERBIRD	ECLIPSE	NETBEANS
Faulty	79.0%	76.7%	25.7%	37.6%
p (SVM)	86.9%	84.9%	73.6%	73.1%
r (SVM)	97.3%	98.2%	64.0%	61.8%
F (SVM)	0.92	0.91	0.69	0.67

VI. DISCUSSION, THREATS TO VALIDITY AND IMPLICATIONS FOR FUTURE WORK

Prior to concluding our article, we discuss a number of limitations of our analysis as well as resulting threats to validity. As described in section III, all our findings are based on interactions recorded in the BUGZILLA installation of the projects MOZILLA FIREFOX, MOZILLA THUNDERBIRD, ECLIPSE and NETBEANS. Clearly, a significant threat to the applicability of our approach for general collaborative software engineering is that we were mainly focused on these four OSS communities. However, we argue that these particular projects represent communities with different levels of heterogeneity with respect to the level of contributions, commitment, technical proficiency and commercial influence by companies. In particular, the communities of MOZILLA FIREFOX and MOZILLA THUNDERBIRD target a rather general audience without particular technical proficiency, while ECLIPSE and NETBEANS are more focused on software developers. As such, our particular choice of communities may be considered as covering different ends of the spectrum of technical proficiency of users. Our analysis shows that, even for such diverse projects, machine learning techniques based on quantitative measures of social embeddedness yield high accuracy results when predicting bug report quality. Therefore our contribution can be seen as a proof of concept case study. Nevertheless,

RIGHTSLINK

we are currently collecting and analyzing data as well as qualitative insights on the social organization of a number of additional communities in order to generalize our results.

Although our analysis focuses on the BUGZILLA communities of OSS projects, our methodology is - in general - not limited to these. Any issue tracking system which records time-stamped direct interactions between its users can be used to extract evolving collaboration networks and thus to compute quantitative measures for social embeddedness. However, whether these measures can be used for highly accurate, automated bug categorization in settings other than the ones studied in this paper (like e.g. commercial software production or collaborations in smaller or less diverse teams) requires further studies and is beyond the scope of our work.

While we have presented a set of quantitative results regarding the relation between the network position of bug reporters and the outcome of bug report processing, it is unclear what are the exact social mechanisms at work. In order to gain a better insight into this question, we have created a survey that was sent to the community managers of the projects considered in this case study. Indeed, in their replies the community managers of ECLIPSE and NETBEANS confirmed that such a relation may exist. Specifically, we received feedback indicating that for the NETBEANS community "one of the criteria developers use while choosing bugs for fixing is reproducible case and/or reputation of the reporter". Similarly, for the ECLIPSE project community managers confirmed that "a committer is often times more likely to spend triage time on a bug from somebody with a known reputation for quality". Unfortunately, we did not receive any feedback to our survey for the communities of MOZILLA FIREFOX and MOZILLA THUNDERBIRD.

For the network measures studied in this paper, we only used the direct dyadic relations *CC* (i.e. users subscribing to receive information about future updates on bug reports) and *Assign* (i.e. users assigning the task of handling a bug to another one). While these recorded interactions are clearly associated with users knowing about and interacting with each other, the resulting network must clearly be seen as a mere proxy for the actual social organization of a community. In particular, in our study of network measures we did not consider further relations that may be extracted for instance from the sequence of comments on a bug. The reason for not considering these is the lower fidelity with respect to whether an extracted relation is really associated with direct communication or collaboration. Furthermore, in our study we so far did not use further potential data sources, like mailing lists or threaded forum communication that could be used to augment our network perspective in a subsequent analysis.

Another remark related to the measures of social embeddedness adopted in our analysis is that they can be quantified right away after a bug report is submitted. As we show in the paper, this works well for OSS communities that have accumulated enough samples to apply machine learning techniques. Therefore the extension of this methodology to newly born communities remains a challenge.

A possible reason of concern is the fact that we use a fixed-size window of 30 days to construct the networks used in our analysis. Although we have obtained high accuracy results for this particular choice of window size, we are further investigating whether tuning this parameter to each community independently will further increase performance.

Finally, the application of machine learning comes at the risk of overfitting data by using a too large fraction of training data. In order to avoid this pitfall, we limited the fraction of randomly chosen training data to 5.0%. To foster the reproducibility of our results and to facilitate the implementation of similar approaches of social awareness in practical support infrastructures, the source code of the SVM classifier (written in the R language) as well as the data sets studied in our analysis are available online[1].

VII. CONCLUSIONS

In this paper we have studied to what extent the positions of bug reporters in the collaboration networks of four OSS communities are indicative for the quality of contributed bug reports. We have addressed this question from the perspective of evolving complex networks that have been extracted from a comprehensive data set on $700,000$ bug reports for the projects MOZILLA FIREFOX, MOZILLA THUNDERBIRD, ECLIPSE and NETBEANS. The main results of our case study on these communities are the following:

(1) We study the evolution of bug reporter centrality in *evolving collaboration networks*, using a time resolution of 30 days over a total period of 10 years. For the project MOZILLA FIREFOX, we are able to validate our hypothesis that the eigenvector centrality of bug reporters increases after the submission of valid bug reports (i.e. reports that refer to actual software bugs, are no duplicates and contain all necessary information). We observe the opposite relation for MOZILLA THUNDERBIRD.

(2) In all projects we were able to validate our hypothesis that there is a statistically significant decrease of eigenvector centrality following the submission of duplicate bugs.

(3) For the projects MOZILLA FIREFOX, MOZILLA THUNDERBIRD and NETBEANS we were able to validate our hypothesis that the eigenvector centrality of users reporting *valid* bug reports is significantly higher than those of users submitting *faulty* bug reports. From this we conclude that the position of bug reporters in the collaboration network of OSS communities is indicative for the quality of bug reports.

(4) Based on this finding, we develop an automated bug report classification mechanism. We use nine topological measures at the level of bug reporters (eigenvector, betweenness and closeness centrality, k-coreness, clustering coefficient, in-, out- and total degree as well as membership in the largest connected component) for the prediction of whether a reported bug is *valid* or *faulty*. Based on a support vector machine and depending on the project considered, our automated classification achieves a precision of up to 90.3% and an F-score of up to 0.92.

[1] see http://www.sg.ethz.ch/research/topics/social-se/data/

RIGHTSLINK

We would like to emphasize the fact that - although it is merely based on measures quantifying the network position of bug reporters - *our proposed classification mechanism achieves a remarkably high accuracy across different communities*. The combination of our approach with further features used in previous studies of automated bug classification is likely to further improve its accuracy. Our case study can thus be seen as a contribution towards classification schemes that are highly accurate, yet simple enough to be of practical relevance in the design of support infrastructures.

ACKNOWLEDGMENT

This work was supported by the SNF through grant CR12I1_125298. We would like to acknowledge the contribution of Emre Sarigöl to the collection and preprocessing of data, and the communities of NETBEANS and ECLIPSE for sharing their insights with us.

REFERENCES

[1] J. Anvik, "Automating bug report assignment," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. ACM, 2006, pp. 937–940.

[2] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*, 2007, pp. 34–43.

[3] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *IEEE International Conference on Software Maintenance*. Ieee, 2010, pp. 1–10.

[4] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K.-i. Matsumoto, "Predicting reopened bugs: A case study on the eclipse project," in *Proceedings of the 2010 17th IEEE Working Conference on Reverse Engineering*, 2010, pp. 249–258.

[5] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing," *29th International Conference on Software Engineering (ICSE'07)*, pp. 499–510, May 2007.

[6] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," *Proceedings of the 13th international conference on Software engineering - ICSE '08*, p. 461, 2008.

[7] D. Cubranic and G. C. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*, 2004, pp. 92–97.

[8] A. Podgurski, D. Leon, and P. Francis, "Automated support for classifying software failure reports," in *Proceedings of the 25th International Conference on Software Engineering, ICSE'03*, 2003, pp. 465–475.

[9] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," in *Proceedings of the Second ACM-IEEE international symposium ESEM*, 2008, pp. 2–11.

[10] P. Guo and T. Zimmermann, "Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows," in *Proceedings of the 32nd ACM/IEEE Conference on Software Engineering*, 2010, pp. 495–504.

[11] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of the 2012 International Conference on Software Engineering, ICSE*, 2012, pp. 25–35.

[12] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," *2012 34th International Conference on Software Engineering (ICSE)*, pp. 1074–1083, Jun. 2012.

[13] 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*, vol. 11, no. 3, pp. 309–346, 2002.

[14] K. Ehrlich and M. Cataldo, "All-for-one and one-for-all?: a multi-level analysis of communication patterns and individual performance in geographically distributed software development," in *Proceedings of the ACM CSCW*, 2012, pp. 945–954.

[15] M. S. Zanetti, E. Sarigöl, I. Scholtes, C. J. Tessone, and F. Schweitzer, "A quantitative study of social organisation in open source software communities," in *ICCSW*, 2012, pp. 116–122.

[16] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" *Proceedings of the 16th ACM SIGSOFT/FSE International Symposium on Foundations of software engineering*, p. 308, 2008.

[17] J. Wang and J. M. Carroll, "Beyond fixing bugs: case studies of creative collaboration in open source software bug fixing processes," in *Proceedings of the 8th ACM conference on Creativity and cognition*, 2011, pp. 397–398.

[18] N. Bettenburg and A. E. Hassan, "Studying the Impact of Social Structures on Software Quality," *2010 IEEE 18th International Conference on Program Comprehension*, pp. 124–133, 2010.

[19] T. Wolf, A. Schroter, and D. Damian, "Mining task-based social networks to explore collaboration in software teams," *Software,*, pp. 58–66, 2009.

[20] T. Wolf and A. Schroter, "Predicting build failures using social network analysis on developer communication," *Proceedings of the 31st*, pp. 1–11, 2009.

[21] G. A. Di Lucca, M. Di Penta, and S. Gradara, "An approach to classify software maintenance requests," in *Software Maintenance, 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 93–102.

[22] K. Somasundaram and G. C. Murphy, "Automatic categorization of bug reports using latent dirichlet allocation," in *Proceedings of the 5th India Software Engineering Conference*, ser. ISEC '12. ACM, 2012, pp. 125–130.

[23] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *proceedings of the 34th ICSE*, 2012, pp. 419–429.

[24] N. Serrano and I. Ciordia, "Bugzilla, itracker, and other bug trackers," *Software, IEEE*, vol. 22, no. 2, pp. 11–13, 2005.

[25] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[26] M. E. J. Newman, *Networks: an introduction*. Oxford Univ Press, 2010.

[27] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1979.

[28] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.

[29] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, 2012.

[30] A. Garas, F. Schweitzer, and S. Havlin, "A k-shell decomposition method for weighted networks," *New Journal of Physics*, vol. 14, no. 8, p. 083030, August 2012.

[31] M. Hollander and D. Wolfe, *Nonparametric statistical methods*. Wiley-Interscience, 1999.

[32] G. Salton and M. McGill, *Introduction to modern information retrieval*. McGraw-Hill, Inc., 1986.

[33] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

RIGHTSLINK