Crowdsourced Bug Triaging

Ali Sajedi Badashian, Abram Hindle, Eleni Stroulia
Department of Computing Science, University of Alberta, Edmonton, Canada
{alisajedi, abram.hindle, stroulia}@ualberta.ca

***Abstract*—Bug triaging and assignment is a time-consuming task in big projects. Most research in this area examines the developers' prior development and bug-fixing activities in order to recognize their areas of expertise and assign to them relevant bug fixes. We propose a novel method that exploits a new source of evidence for the developers' expertise, namely their contributions to Q&A platforms such as Stack Overflow. We evaluated this method in the context of the 20 largest GitHub projects, considering 7144 bug reports. Our results demonstrate that our method exhibits superior accuracy to other state-of-the-art methods, and that future bug-assignment algorithms should consider exploring other sources of expertise, beyond the project's version-control system and bug tracker.**

I. Introduction and Background

Bug-triaging-and-assignment has received substantial attention by the software-engineering community [1], [2], [3], [4], [5], [6], [7], [8], [9]. Given a bug report, the goal is to select and rank relevant project developers, who would have the relevant knowledge to fix it. This problem touches on two relevant research areas: (a) expertise identification and recommendation, and (b) bug triaging.

Relevant to *expertise-recommendation*, Venkataramani et al. [10] described a system for recommending Stack Overflow members qualified to answer specific questions. The system considers the names of the classes and methods to which the developers have contributed to infer their expertise. Similarly, Fritz et al. [11] posed the "Degree of Knowledge" (DOK) metric to determine the level of a developer's knowledge regarding a code element –class, method or field– based on the developer's contribution to the development of this element. Mockus and Herbsleb [12] described "Expertise Browser" (EB), a tool that identifies the developers' expertise about code and documentation, considering system commits and changes to classes, sub-systems, packages, etc. Teyton et al. [13] developed XTic, a system taht requires as input a set of skills of interest and provides an automatic process that extracts skills and experience levels from source code repositories. Zhang et al. [14] described a method for constructing a "Community Expertise Network" (CEN), from the post-reply relations of Java Forum users. Assuming that asking questions is evidence of ignorance and providing answers is evidence of expertise, they defined and demonstrated the usefulnes of the Z-score as an expertise indicator: $Z = (a - q)/\sqrt{(a + q)}$, where q and a are respectively the number of questions asked and answered by a community member.

There has already been substantial research on *bug triaging*, which has produced a number of different techniques to select the (*top-k*, where k is typically 1 and 5) most capable developer(s) to resolve a given bug report. Table I summarizes some key results of this work. In relation to this earlier research, the method we propose in this paper is unique in that it uses the developer's expertise, as demonstrated by the developer's contributions to Stack Overflow, as a source of evidence regarding the competence of a developer to fix a bug. We describe our method in Section II and we report on our experimental-evaluation results in Section III. Reflecting on these results and the corresponding results of earlier studies, we argue for the merits of our method in Section III-C. Finally, in Section IV, we conclude with the lessons we hope to share with the community and our plans for future work.

II. A Social Bug-Triaging Method

Motivated by the overlap in the activities of developers in GitHub and Stack Overflow [20], [21], in this work, we ask *what if we combine expertise-recommendation based on networks like Stack Overflow with triaging of issues?* and we describe a method that exploits evidence of expertise in the developers' Stack Overflow activity traces, for identifying candidate bug fixers in GitHub. Let us describe the key intuition of our method with a simple example: if a developer has answered several questions tagged with the *jquery* keyword, and her answers have received the community's approval with many upVotes, she has a "proven" expertise record in *jquery*; therefore, she should be a likely candidate for fixing bugs whose description includes the *jquery* keyword.

Consider, for example, the activity around five Stack Overflow questions, shown in Table II. The bold tags indicate keywords that also appear in the bug report, which needs to be addressed. The middle section of Table II reports the simple *AnswerNum* score, namely the total number of questions answered by the developer, and the developer's *Z-score*[14] as described above. These two expertise indicators completely ignore the bug at hand, which is why we developed the three additional scores, reported at the bottom of Table II and described in detail in Section II-A below. We use Stack Overflow tags for cross-referencing Stack Overflow and GitHub. As a sanity check for the applicability of tags in GitHub bug reports, we examined the bug reports in 3 selected projects (out of the 20 projects considered in this study) and found that the textual information of each bug report (including *project language*, *project description*, *issue title* and *issue body*) mentions between 2 to 89 tags (avg=14.9 and σ=11.5). In effect, tags are keywords, curated by the community, that define a common set of vocabularies for developers to exchange information without the need for stop-word and noise-word removal from the bug-report texts.

A. A Bug-Specific Social Metric of Expertise

Given a bug report, *b*, the objective is to estimate a developer's expertise and potential ability to fix it. To that end, we define **matched_tags**$_{q,b}$ and **matched_tags**$_{a,b}$, as the set of tags of a specific question (q) and its answers (a) that

TABLE I
RECENT BUG-TRIAGING METHODS

Authors	Basic method / Information used	Effectiveness
Čubranić and Murphy[2]	Naive Bayes classification of bug reports (i.e., "text documents") to developers (i.e., "classes"); uses bug summary and description.	up to 30% *top-1*
Anvik et al. [1]	Support Vector Machine (SVM) classification of bug reports (i.e., "text documents") to developers (i.e., "classes"); uses bug summary and description.	up to 57%, 64% and 18% *top-3* accuracy
Tamrawi et al. [15]	A fuzzy-set representation of the relations between developers and the bug reports' technical terms; uses bug summary and description.	average 40% and 75% for *top-1* and *top-5* accuracy over 7 projects
Lamkanfi et al. [16]	Multinomial Naive Bayes and some other ML approaches; uses bug report summary, description, severity and component.	79% accuracy in predicting severity of bug reports
Lin et al. [17]	SVM and C4.5 classifiers; uses bug summary and description, type, class, priority, submitter and the module ID.	up to 77% top-10 accuracy
Nguyen et al. [6]	Regression model based on LDA topic modeling; uses bug description.	Just estimated the time to fix for each developer $\pm$2.3 days
Canfora and Cerulo [18]	A probabilistic IR method to query the new bug report's text and find the best developer (considered as a document); uses descriptions of the change requests.	62% and 85% accuracy in two projects
Matter et al. [5]	Vector Space Model (an IR method); uses source-code commits and the bug-report keywords;	34% and 71% *top-1* and *top-10* accuracies
Linares-Vásquez et al. [4]	IR-based concept location techniques; uses text of a change request and source code files.	85% *top-5* accuracy
Shokripour et al. [19]	A method based on information extraction; uses bug summary and description, detailed source code info (comments, names of classes, methods, fields, etc.).	48% and 48% *top-1* and 60% and 89% *top-5* accuracies on two projects (between 57 and 9 developers respectively)
Jeong et al. [3]	Introduced "tossing graphs" of developers to reduce bug reassignment; uses bug report title and description	up to 77% *top-5* accuracy

appear in the textual information of the bug report (*b*). These metrics are calculated for each pairwise combination of bug reports and questions (and answers) provided by the project developers.

We next define $A_{u,b}$, relative weighted answers, and $Q_{u,b}$, relevant weighted questions, to replace *a* and *q* respectively in the original definition of *Z-score*, taking into account the community's assessment of the "quality" of a developer's contributions. The definition of $A_{u,b}$ is shown in Equation 1. At any point in time, for every answer a developer has contributed in the past that is relevant to the bug under consideration, the number of $matched\_tags_{a,b}$ is multiplied with the number of the answer's upVotes (plus one, for the answer itself).

$$A_{u,b} = \sum_{\substack{a \in answers \\ posted\ by\ u}} (upVotes_a + 1) \cdot (matched\_tags_{a,b}) \quad (1)$$

The $Q_{u,b}$ is calculated as shown in Equation 2. In principle, questions are considered as evidence of lack of expertise [14]. However, to mitigate the adverse effects of asking "good" questions, we divide $matched\_tags_{a,b}$ by the number of upVotes (plus one for the question itself). This tends to make the value of $Q_{u,b}$ very small, in comparison to $A_{u,b}$, which is why we use the μ *normalization factor* to adjust it.

$$Q_{u,b} = \mu \cdot \sum_{\substack{q \in questions \\ posted\ by\ u}} \frac{(matched\_tags_{q,b})}{(upVotes_q + 1)} \quad (2)$$

The *Social Subject-Aware Z-score* (*SSA_Z-score*) can then be defined, as shown in Equation 3.

$$SSA\_Z\text{-}score_{u,b} = \frac{(A_{u,b} - Q_{u,b})}{\sqrt{(A_{u,b} + Q_{u,b})}} \quad (3)$$

Note that, in order to capture a temporally-aware measure of a developer's expertise, this formula only involves questions and answers posted before the time when the bug was reported (i.e., $t_q < t_b$ and $t_a < t_b$).

B. A Recency-Sensitive SSA_Z-score

Developers' expertise shifts over time as they work on different projects with potentially different technologies. This is why, many related expertise-modeling methdologies [5], [19], [1] include a decay factor for older evidence of expertise and weigh it less than more recent one. To capture this intuition, that "recent evidence of expertise is more valuable", we defined *Recency_of_activity* as shown in Equation 4 below.

$$Recency\_of\_activity_{u,b} = \sum_{\substack{i \in \\ bugs\ of\ u}} \frac{1}{1 + |\{d | d \in bugs\ of\ u \wedge t_d > t_i \wedge t_d < t_b\}|} \quad (4)$$

Note that *t* is the time that the bug report was submitted, and the denominator counts the number of bug reports that occurred between *i* and *b*. The intuition here is that bug-fixing exhibits locality, namely that "the developers that have been fixing bugs recently are likely to fix bug reports in the near future" [15]. We combine the two last metrics to assign each user a new score regarding the bug currently being triaged:

$$Triage\_score_{u,b} = \alpha \cdot (SSA\_Z\text{-}score_{u,b}) + \beta \cdot (Recency\_of\_activity_{u,b}) \quad (5)$$

In this formula, α and β are parameters, which are tuned following a systematic process explained in Section III-B. Having the *Triage_score* for all users in the community over a bug report, our bug-triaging algorithm sorts the users and reports the top *k* developers to fix the bug.

III. EVALUATION

We obtained two Stack Overflow data sets [22], [23] (approximately 65GB and 90GB). They consist of several XML files including information about 2,332,403 and 3,080,577 users, their posts, tags, votes, etc.). In order to link these users to GitHub, their *email hash* is needed [21], which is provided by the older data set. We merged these two data sets to get a large data set including users of old data set with newer posts.

TABLE II
EXAMPLE OF DIFFERENT SCORES FOR USERS: EACH ROW SHOWS A QUESTION AND NUMBER OF UPVOTES FOR EACH OF ITS ANSWERS (BY DIFFERENT USERS)

Questions \ Answerer	Bob	Ali	Taylor	Yakob	Jane	Brian	Harpreet
Q1 by Yakob, 3 upVotes tags: [version control], [**open source**]	46	5	53		28		
Q2 by Jane 1 upVotes tags: [ajax], [**data**], [**search**], [jquery], [php]	20	16	22	6			
Q3 by Yakob, 21 upVotes tags: [**lucene**], [**elasticsearch**], [php], [**java**]	11	14	29		10		
Q4 by Ali, 0 upVotes tags: [**https**], [css], [**java**], [**jira**], [**data**]	27			0		86	
Q5 by Harpreet, 70 upVotes tags: [**java**], [**ajax**], [**https**], [**xml**], [**lucene**]	1	18	42	-4	14	98	
AnswerNum	5	4	4	3	3	2	0
Z-score	2.24	1.34	2	0.45	1	1.41	-1
A	$\frac{(46+1)\cdot1 + (20+1)\cdot2 + (11+1)\cdot3 + (27+1)\cdot4 + (1+1)\cdot5}{=247}$	$\frac{(5+1)\cdot1 + (16+1)\cdot2 + (14+1)\cdot3 + (18+1)\cdot5}{=180}$	$\frac{(53+1)\cdot1 + (22+1)\cdot2 + (29+1)\cdot3 + (42+1)\cdot5}{=405}$	$\frac{(6+1)\cdot2 + (0+1)\cdot4 + (-4+1)\cdot5}{=3}$	$\frac{(28+1)\cdot1 + (10+1)\cdot3 + (14+1)\cdot5}{=137}$	$\frac{(86+1)\cdot4 + (98+1)\cdot5}{=843}$	0
Q ($\mu = 20$)	0	$20\cdot(\frac{4}{0+1})$ $= 80$	0	$20\cdot(\frac{1}{3+1}+\frac{3}{21+1})$=7.7	$20\cdot(\frac{2}{1+1})$ $= 20$	0	$20\cdot(\frac{5}{70+1})$ $= 1.41$
SSA_Z-score	15.72	6.20	20.12	-1.44	9.34	29.03	-1.19

We used the GHTorrent mySQL dump [24] (with a size of about 21GB) containing information about 4,212,377 GitHub users and their project memberships. However, this data set did not include the textual information of the bug reports. We obtained this information from a set of MongoDB dumps from the GHTorrent site (210GB) including information of 2,908,292 users. Again, we merged the two data sets and obtained a large data set including information about GitHub users, projects and bug reports. Both the GitHub and Stack Overflow data sets include information of the users and their activities from 2008 to 2014. As our method assigns bugs to developers with a presence in both GitHub and Stack Overflow, we encoded users's emails in GitHub with MD-5 function and compared them with every e-mail hash available in Stack Overflow [20], [21]. With this approach, we found 358,472 common users[1].

A. Experiment Setup and Implementation

We first extracted the *community members* of each project as the union of the sets of project members, committers, bug reporters, and bug assignees. We then refined this community to include only developers who had posted some questions or answers on Stack Overflow. Next, we identified the top 20 ranked projects based on the number of their community members [1]

For the selected 20 projects, the number of community members vary from 28 to 822 (average=127, σ=169, median=87). Out of 14,172 bug reports in all the selected projects, we examined 7144 bug reports that have been assigned to one of the community members in the related project. Note that we could not use the rest of bug reports since they were assigned to developers with no Stack Overflow activity. However, if this approach is applied in the workplace, alternative networks should be tested and used. We used bug reports from three of these projects for training and tuning purposes and 17 for final evaluation. For each bug report in each project, we ran our algorithm to compute the expertise score of all project-community members and ranked the users from the highest score to the lowest. We report *top-1* and *top-5* accuracies as well as Mean Average Precision (MAP) as a synthesized, rank-based evaluation measure [25].

To compare with other state-of-the-art methods, we experimented with the `scikit-learn` [2] implementations of a number of machine-learning algorithms used as the basis for the above research [1], [2], [16], [17] which we applied[3] to our own data set: (1) 1NN, 3NN and 5NN; (2) Naive Bayes; (3) Multinomial Naive Bayes; and (4) SVM.

We used the words in the title and description of the bug reports as the TFIDF feature vectors. We developed an online train-and-test method; train them on first n-1 bug reports and then test on the n^{th}. Then recursively train on first n bug reports and test on $n+1^{th}$.

We used the following parameters for `scikit-learn` machine learners. For KNN, we chose k as the parameter (1, 3 or 5), weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski' and metric_params=None. For Multinomial Naive Bayes, we used Laplace smoothing priors ($\alpha = 1.0$) fit to prior distribution using OneVsRestClassifier classifier strategy. Similarly for Naive Bayes, but it uses multiclass classification. For SVM, we used Support Vector Classification (SVC). We chose RBF kernel, used shrinking heuristic, with gamma kernel coefficient $1/n$ for n features, error penalty=1 and probability=true.

[1]Due to space limitation, our data sets, information of 3+17 projects, Java implementation of our approach and output and tuning results are available online at: http://github.com/alisajedi/BugTriaging

[2]http://scikit-learn.org/stable/

[3] Our Python implementations: http://github.com/abramhindle/bug-triager-scikit/blob/ali/dumpbayes.py More explanation of the ML methods are also available at the repository.

B. Results

Out of 20 projects, we selected three projects (including 490 bug reports). We then measured the performance metrics of different approaches from *AnswerNum* to original Z-score, to Subject-Aware Z-score (*SA_Z-score*), to *Social Subject-Aware Z-score* (*SSA_Z-score*), to the final recency-aware *SSA_Z-score* (*Triage_score*). In each step, we observed an improvement in accuracy. This validates our effort toward considering Stack Overflow upVotes while being aware of bug content. For the purpose of tuning and calibrating our method, we needed to determine the best values for μ, α and β (Equations 2 and 5). The best obtained values are as follows: μ (normalization factor) is set to *"Harmonic Mean plus 1"*, β=1 and α=0.01. The reason for small α value can be because of very large numbers attained for *Social_Z-score* (i.e., number of upVotes multiplied by number of tags, summed over all answers of each user). We apply the parameter values (μ, α and β) obtained from the three test projects into the remaining 17 projects in our final evaluation.

As the final evaluation, we ran our algorithm over 17 projects (holding out the three projects used for tuning) including 6654 bug reports and sorted the recommended developers for each bug report. The average *top-k* accuracies of our approach for k from 1 to 5 are 45.17%, 66.41%, 77.50%, 84.79% and 89.43% respectively. We also obtained a Mean Average Precision (MAP) of 0.633, which is very strong and shows that the harmonic mean of the real assignee is 1.58 over all the bug reports. Note that in bug triaging, MAP is equal to Mean Reciprocal Rank (MRR) of the real assignee over all the recommendations.

We also implemented the other approaches discussed in Section III-A. We ran those experiments to compare the results of our method with other approaches on the same data set. The results for average *top-1* and *top-5* accuracies as well as MAP are shown in Table III.

The values reported in Table III are averages over all 17 projects examined. However, we examined the detailed results for each project and found them close to the mean (median=90.19 and σ=7.81 for *top-5* accuracies). Our results demonstrate that our *Triage_score*, relying on evidence of developers' expertise from their Stack Overflow activities, is very effective in selecting the right assignee for the right bug, much more so than all competing machine-learning algorithms relying exclusively on GitHub data. It is noteworthy that our approach is fast and efficient enough since it avoids the typical text pre-processing of most IR-based methods, such as stemming and indexing. Each bug report was triaged in almost a second, which is fast enough for real-time use.

C. Analysis

The results in Table III demonstrate that our method exhibits the best performance, outperforming all other machine-learning methods in terms of *top-5* accuracy and MAP. 3NN, 5NN and SVM do well for top-1 accuracy, slightly better than our approach. Our average *top-5* accuracy is between 8 to 19 percent better than other approaches. The MAP value of our approach —0.633— corresponds to the harmonic mean 1.58 for the rank of the real assignee (implying that the real assignee frequently appeared in the rank-1 and rank-2 positions in the results). MAP varies from 0.575 (for 1NN) to 0.617 (for SVM as the best approach after our's). Comparing the different algorithms on the same data set demonstrates in the usefulness of our method.

We also compared our results with previously published results. In short, as one of the best obtained accuracies in the previous studies, Shokripour et al. [19] obtained 48% *top-1* and 60% and 89% *top-5* accuracies on two projects (with 57 and 9 developers respectively). Our *top-5* accuracy outperforms theirs, but their approach performs 3% better on *top-1*. Note that their best results were obtained in a project with only 9 candidate developers (our projects included between 28 and 822 developers). Also note that their approach was tested only on 80 and 85 bug reports, as opposed to our 7144 bug reports, which constitutes strong evidence on the robustness of our approach.

To summarize our comparison findings, it is important to mention the following three key points. Our evaluation of our metric is the most thorough reported in the literature (with 20 projects and 7144 bug reports). Our metric highly outperforms all previously reported methods in terms of average *top-5* accuracy, and most of them in terms of average *top-1* accuracy. More importantly, our metric exhibits the highest MAP/MRR.

D. Limitations and Threats to Validity

An external validity threat is that the common users (between Stack Overflow and GitHub) constitute up to 20% of the total number of users in each of these networks. Currently, for privacy reasons, much of the Q&A content at the software social networks is provided anonymously. However, the large number (i.e., thousands) of developers and bug reports on which we tested our approach mitigates this limitation. One could envision that project managers could easily request their developers to provide their IDs in Q&A networks like Stack Overflow, as part of their CV. As a result, more extensive Q&A contributions (or alternative sources of information) will become available to the bug-assignment process.

Another concern is how to treat the phenomenon of developers answering their own questions to announce a commonly encountered issue with some API, library, etc. However, we investigated the questions and answers of members of three projects out of 20 and found that only 3% of their answers are answers to one's own question, and only in around half of these cases, the question is upVoted, meaning that the case did not indicate expertise, but lack of expertise (as we assumed).

IV. Conclusions and Future Work

In this paper, we described a method that effectively utilizes the expertise networks, such as Stack Overflow contributions of developers and their previous bug-assignment history to decide the best candidate developer for fixing a bug. We have thoroughly evaluated our method with 20 popular GitHub projects, comparing its performance (a) against six traditional machine-learning approaches that have been widely used for bug assignment before, and (b) against the reported accuracies of previous bug-triaging publications. Our approach outperforms the competition.

TABLE III
ACCURACY RESULTS FOR DIFFERENT SIMULATED APPROACHES COMPARED WITH OURS

Evaluation Measure \ Method	1NN	3NN	5NN	Naive Bayes	Multinomial Naive Bayes	SVM	Our approach
Top 1 Accuracy (%)	43.09	**46.48**	45.60	43.77	42.75	45.46	45.17
Top 5 Accuracy (%)	70.46	75.63	75.00	78.98	75.97	81.82	**89.43**
MAP	0.575	0.610	0.596	0.609	0.606	0.617	**0.633**

The fundamental novelty of our work is that it takes advantage of contributions of the developers in software Q&A networks as a rich, unexploited socio-technical source of expertise information, beyond their code. This leads us to a more interesting insight: applying various **third-party expertise networks** in bug triaging envisions new horizons for software maintenance community. In fact, the socio-technical information available on the web is a great source of expertise. While some developers contribute in Stack Overflow, many others may prefer Java Forum, Ask Ubuntu, Experts Exchange, Code Project, Web developer, SUN Forums, MSDN Forums and so on. As part of their development process, developers may provide their IDs in their desired software social platforms, to better inform our method regarding their expertise and thus improve the triaging process.

In the future, we plan to handle tag synonyms. Different synonym tags can be integrated in their primary definition addressing in bug reports and Q&A contents. Finally, capturing level of similarity of the keywords in bug reports with tags (e.g., "xml-parser", "xml parser", "xmlparsing" and "xml parsing" compared to tag "xml-parsing") can be a useful extension. Curation, noise reduction, and tag recommender approaches [26] may also be useful in this case.

ACKNOWLEDGMENTS

This work has been partially funded by IBM, the Natural Sciences and Engineering Research Council of Canada (NSERC) and the GRAND NCE.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. ACM, 2006, pp. 361–370.

[2] D. Čubranić and G. C. Murphy, "Automatic bug triage using text categorization," in *In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

[3] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. ACM, 2009, pp. 111–120.

[4] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 451–460.

[5] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, May 2009, pp. 131–140.

[6] T. T. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Topic-based, time-aware bug assignment," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 1, pp. 1–4, Feb. 2014.

[7] M. K. Hossen, H. Kagdi, and D. Poshyvanyk, "Amalgamating source code authors, maintainers, and change proneness to triage change requests," in *Proceedings of the 22nd ICPC*. ACM, 2014, pp. 130–141.

[8] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software: Evolution and Process*, vol. 24, no. 1, pp. 3–33, 2012.

[9] M. Bahrami Zanjani, K. Huzefa, and C. Bird, "Using Developer-Interaction Trails to Triage Change Requests," in *MSR*. IEEE, 2015.

[10] R. Venkataramani, A. Gupta, A. Asadullah, B. Muddu, and V. Bhat, "Discovery of technical expertise from open source code repositories," in *Proceedings of the 22Nd International Conference on World Wide Web Companion*, ser. WWW '13 Companion. International World Wide Web Conferences Steering Committee, 2013, pp. 97–98.

[11] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. ACM, 2010, pp. 385–394.

[12] A. Mockus and J. D. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. ACM, 2002.

[13] C. Teyton, M. Palyart, J.-R. Falleri, F. Morandat, and X. Blanc, "Automatic extraction of developer expertise," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014, p. 8.

[14] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: Structure and algorithms," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. ACM, 2007, pp. 221–230.

[15] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. ACM, 2011, pp. 884–887.

[16] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. IEEE, 2011, pp. 249–258.

[17] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using chinese bug data," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. IEEE Computer Society, 2009.

[18] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, ser. SAC '06. ACM, 2006, pp. 1767–1772.

[19] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. IEEE Press, 2013, pp. 2–11.

[20] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: associations between software development and crowdsourced knowledge," in *Social Computing (SocialCom), 2013 International Conference on*. IEEE, 2013, pp. 188–195.

[21] A. Sajedi, A. Esteki, A. GholiPour, A. Hindle, and E. Stroulia, "Involvement, contribution and influence in github and stack overflow," in *Proceedings of the 2014 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '14. Markham, Toronto, Canada: ACM, 2014.

[22] Stack Exchange Community. Is there a direct download link with a raw data dump of stack overflow? "http://meta.stackexchange.com/questions/198915/is-there-a-direct-download-link-with-a-raw-data-dump-of-stack-overflow-not-a-t", Visited on 2014/08/20.

[23] Stack Exchange, Inc. Stack exchange data dump. "https://archive.org/details/stackexchange", Visited on 2014/08/20.

[24] G. Gousios, "The ghtorent dataset and tool suite," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 233–236.

[25] C.-P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 181–190.

[26] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "Entagrec: an enhanced tag recommendation system for software information sites," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 291–300.