Crowdsourced Bug Triaging: Leveraging Q&A Platforms for Bug Assignment

No Author Given

No Institute Given

Abstract. Bug triaging, *i.e.,* assigning a bug report to the "best" person to address it, involves identifying a list of developers that are qualified to understand and address the bug report, and then ranking them according to their expertise. Most research in this area examines the description of the bug report and the developers' prior development and bug-fixing activities. In this paper, we propose a novel method that exploits a new source of evidence for the developers' expertise, namely their contributions in *Stack Overflow*, the popular software Question and Answer (Q&A) platform. The key intuition of our method is that the questions a developer asks and answers in *Stack Overflow*, or more generally in software Q&A platforms, can potentially be an excellent indicator of his/her expertise. Motivated by this idea, our method uses the bug-report description as a guide for selecting relevant *Stack Overflow* contributions on the basis of which to identify developers with the necessary expertise to close the bug under examination. 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.

1 Introduction

Software development, today more than ever, is a community-of-practice activity. Developers often work on multiple projects, hosted on large-scale software repository platforms, such as *GitHub* and *BitBucket.* They access and contribute information to open question-answering web sites, such as *Java Forum*, *Yahoo! Answers* and *Stack Overflow* [1]. Through the developers' participation on these code-sharing and question-answering platforms, rich evidence of their software development expertise is collected. Understanding the developers' expertise is relevant to many software-engineering activities, including "onboarding" of new project members so that their expertise is best utilized in the new context, forming new teams that have the necessary expertise to take on new projects, and bug triaging and assignment to the person that is best skilled to fix it.

In this paper we focus on the bug-triaging-and-assignment task, which has already received substantial attention by the software-engineering community [2][8][10][14][15][18]. The typical formulation of **bug triaging** problem aims at

[1] `http://www.coderanch.com/forums`, `http://answers.yahoo.com`, and `http://stackoverflow.com/`

ranking a number of developers that could potentially fix a given bug report. Most solutions to date have considered developers' expertise, using their past development and bug-resolving contribution as evidence. In contrast, we describe and report on the effectiveness of a bug-assignment method that uses expertise networks extracted from social software-development platforms.

At a high level, our work makes two novel contributions to the bug-triaging research. First, we demonstrate that as a software focused Q&A web site, *Stack Overflow* contains valuable information about the expertise of the participating developers, which may be exploited to support bug triaging. Second, we comparatively investigate a family of methods for analyzing *Stack Overflow* posts to precisely understand how to improve state-of-the-art bug-triaging methods.

The rest of the paper is as follows. Section 2 sets the background context for our work. Section 3 describes in detail our new bug-assignment method, ranking the expertise of developers based on a new metric relying on *Stack Overflow*. Section 4 reports on the evaluation of our method. Finally, Section 6 concludes with a summary of the take-home lessons of this work.

2 Literature Review

There are two categories of previous research relevant to this body of work: (a) expertise identification and recommendation; and (b) bug triaging.

Expertise Identification and Recommendation Venkaratamani *et al.* [30] described a system for recommending specific questions to *Stack Overflow* members qualified to answer them. The system infers the developers' expertise based on the names of the classes and methods to which the developers have contributed. Similarly, Fritz *et al.* [9] developed 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 [16] developed the Expertise Browser (EB), a tool that identifies the developers' expertise from their code and documentation, considering system commits and changes to classes, sub-systems, packages, etc.

Zhang *et al.* [31] described a method for constructing a "Community Expertise Network" (CEN) from the post-reply relations of Java Forum users. They then ranked the users' expertise using the PageRank [6] and HITS (Hyperlink-Induced Topic Search) [11] algorithms on this network.

Bug Triaging Previous research in bug-triaging has produced a number of different techniques for selecting the (list of k) most capable developer(s) to resolve a given bug report. Typically the first developer in the list is selected as the bug assignee but, if this developer is unavailable or somehow unsuitable to work on the bug report, the other developers in the recommendations list may be tasked with the bug. Given this problem formulation, most researchers evaluate their methods by reporting *top-k* "accuracy" [8,27,28,1,13,12,26,10,5] (hit ratio in the *top-k* recommended list) or precision-and-recall [2,3,1,7,15,25] (precision is the percentage of the suggested developers who were actual bug fixers and recall is the percentage of bug fixers who were actually suggested).

Machine Learning (ML) approaches: Čubranić and Murphy[8] used a Naive Bayes classifier to assign each bug report (a "text document" consisting of the bug summary and description) to a developer (seen as the "class"). Their classifier was able to predict the bug assignee with a *top-1* accuracy of up to 30%.

Next, Anvik *et al.* [2] proposed a Support Vector Machine (SVM) method as a more effective text classifier for this problem, reporting up to 57%, 64% and 18% *top-3* accuracy. Additionally considering the bug-report severity and priority [3] resulted in 75%, 70%, 84%, 98% and 98% *top-5* accuracy. Note that the last two high-accuracy results are were obtained in very small projects, with 6 and 11 developers respectively. A subsequent method, taking also into account information about the components linked to bugs and the list of active developers resulted in 64% and 86% accuracies in two projects [1].

Lin *et al.* [13] used SVM and C4.5 classifiers, considering the bug-report textual data (title and description) as well as the bug type, class, priority, submitter and the module IDs, and obtained up to 77% accuracy.

Considering severity and component of the bug reports in addition to the textual descriptions, Lamkanfi *et al.* [12] compared the effectiveness of four ML approaches, Naive Bayes, Multinomial Naive Bayes, 1NN and SVM in predicting the real assignee. They reported Multinomial Naive Bayes as the most accurate method with 79% accuracy.

Naguib *et al.* [17] used LDA to assign the bug reports to topics. Then, mining the activity profiles of the developers in a bug-tracking repository, they associate topics to developers. Finally, they suggest the developers with the most topics matching with the bug-report topics. They obtained up to 75% *top-5* accuracy.
Information Retrieval (IR) approaches: Canfora and Cerulo [7] consider each developer as a document by aggregating the textual descriptions of the change requests that the developer has addressed. Given a new bug report, the textual description of the new request is used as a query to the document repository to retrieve the candidate developer. This method achieved 62% and 85% accuracy in two projects.

Develect, by Matter *et al.* [15], employs the Vector Space Model (VSM) and relies on a vocabulary of "technical terms" collected from the developers' source-code commits and the bug-report keywords. The developer's expertise is modeled as a term vector, based on that developer's commit history. Given a new bug report, the closest –according to the cosine distance– developer is identified. This method achieved up to 34% and 71% *top-1* and *top-10* accuracies.

Linares-Vásquez *et al.* [14] applied IR-based concept-location techniques [19] to locate the source code files relevant to the text-change request. Source-code authorship information of these files was used to recommend expert developers and they obtained up to 65% precision.

Shokripour *et al.* [26] proposed an assignee recommender for the bug reports based on information extracted from the developers' source code, comments, previously fixed bugs, and source code change locations. A subsequent study [25] improved these results using additional data, such as the source-code files, commits and comments of the developers, names of classes, methods, fields and parameters in the source code. The maximum *top-5* accuracy of their approach on three different projects was 62%. They obtained 48% and 48% *top-1* and

60% and 89% *top-5* accuracies on two projects (between 57 and 9 developers respectively).

Other approaches: Tamrawi *et al.* [27] introduced a fuzzy approach that computes a score for each "developer - technical term" based on the technical terms available in previous bug reports and their fixing history by the developers. Considering the new bug report, they calculate a score for each developer as a candidate assignee by combining his/her scores for all the technical terms associated with the bug report in question. This method was shown to achieve between b40% and 75% for *top-1* and *top-5* accuracy over 7 projects.

A number of studies have examined bug reassignments, the reasons that cause them, and ways to reduce them [4] [32]. To reduce bug reassignments, Jeong *et al.* [10] introduced "tossing graphs" of developers (as nodes) and edges between them, weighed by the number of times the destination developer was assigned a bug originally assigned to the source developer. Then, beginning with the first prediction (developer candidate) in hand, they used this graph to predict the next developer by consulting this graph. They obtained up to 77% *top-5* accuracy.

All the above studies some combination of the bug textual and categorical attributes, the bug code components, and the developers' coding and bug-fixing contributions. Our method is unique in that it uses the developers' *Stack Overflow* questions and answers, as well as their previous bug assignments, and correlates these contributions to relevant bugs based on the semantic tags they share.

3 A Social Bug-Triaging Model

Software developers today contribute to a variety of social platforms, including social software-development platforms, question-and-answering communities, technical blogs, and presentation-sharing web sites. The key intuition of our work is that these contributions constitute evidence of expertise that can be exploited in the context of bug triaging. More specifically, in this paper, we analyze the developers' contributions in *Stack Overflow* for assigning them to *GitHub* bug reports. Focusing on the overlap of the two social platforms [24], our approach examines the questions and answers in *Stack Overflow* that pertain to the terms mentioned in a bug report's title and description. It uses *Stack Overflow* tags for cross-referencing *GitHub* bug reports with *Stack Overflow* questions and answers. Tags categorize the questions and their corresponding answers in terms of a few well-known technical terms. The community curates these tags to improve their quality: the person asking a question selects the initial tags for the question (out of around 40,000 available but evolving tags) and expert community members, who enjoy a reputation above some threshold, can edit them. Tags are also used as indication of expertise; for example, the person answering a question tagged with *Android* and *Java* is assumed to be knowledgeable in these two domains. Furthermore, the more upVotes this answers collects, the more knowledgeable this answerer is assumed to be.

Figure 1 summarizes the elements of interest in a real bug report[2]. Some of the words in the bug-report's title and description are shown as *italic* because they also appear as tags in the *Stack Overflow* questions reported in Table 1, where they are shown in bold.

Bug report title: TooManyOpenFiles might cause data-loss in *ElasticSearch Lucene*
Bug report body: Under certain circumstances a TooManyOpenFiles exception in *Java* thrown as FileNotFoundException might cause *data* loss where entire shards *lucene* indices are deleted. This is mainly caused by Lucene-4870 *https* issues.apache.org *jira* browse LUCENE-4870 - currently all *Elasticsearch* releases are affected by this.
Project title: *elasticSearch*
Project description: *Open Source* Distributed RESTful *Search* Engine
Project language: *Java*

Fig. 1. An example bug report (selected fields)

Table 1 reports partial information about five questions in *Stack Overflow* and the answers provided by seven developers. Each question is associated with the developer who asked it, the number of upVotes it received, and its thematic tags. The questions are sorted based on the number of their tags that match with the bug-report textual information (in Figure 1) and are shown in bold under each question. The more tags the question shares with the bug-report terms, the more relevant it is to the bug report. We will use these tags to characterize the *expertise areas* required to address the bug report in question.

3.1 Social Metrics of Expertise

Zhang *et al.* [31] introduced a family of metrics for measuring expertise in social networks. The simplest one is *AnswerNum*, the number of answers contributed by a user. However, while answering a question is an indication of expertise, asking a question is an indication of lack of expertise. Z_score is a more sophisticated metric that considers both questions and answers: $Z = (a-q)/\sqrt{(a+q)}$. In this formula, q and a are the numbers of the questions and answers correspondingly posted by user u. If a user asks as many questions as he answers, his Z_score will be close to 0. Developers who answer more questions than they ask have positive Z_scores, and vice versa. The *Z_score* is undefined for users who have not asked nor answered a question. The developers in Table 1 are ordered (left to right) in descending *AnswerNum* order.

3.2 A Bug-Specific Social Metric of Expertise

The *Z_score* would likely identify the most active question answerers as the preferred bug assignees every time, consistently ignoring all other developers. To prevent this phenomenon, we have chosen to refine the *Z_score* with bug-specific information. As discussed before, we use *Stack Overflow* tags as a cross-referencing mechanism between *GitHub* bug reports and *Stack Overflow* questions and answers. Developers facing problems with their tasks, use these tags,

[2] `https://github.com/elasticsearch/elasticsearch/issues/2812` 2014-08-20

Table 1. The activity of some developers in *Stack Overflow*, and their various expertise scores.

Question/Answerer	up Votes	Bob	Ali	Joe	Mike	Jane	Tom	Ben
Q1/Mike; version control, **open source**	3	46	5	53		28		
Q2/Jane; ajax, **data**, **search**, jquery, php	1	20	16	22	6			
Q3/Mike; **elasticsearch**, php, **java**, **lucene**	21	11	14	29		10		
Q4/Ali; **https**, css, **java**, **jira**, **data**	0	27			0		86	
Q5/Ben; **search**,**java**,**lucene**, **elasticsearch**,**https**	70	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_score		$\frac{(46+1).1+(20+1).2+(11+1).3+(27+1).4+(1+1).5}{247}$ =	$\frac{(5+1).1+(16+1).2+(14+1).3+(18+1).5=}{180}$	$\frac{(53+1).1+(22+1).2+(29+1).3+(42+1).5=}{405}$	$\frac{(6+1).2+(0+1).4+(-4+1).5=}{3}$	$\frac{(28+1).1+(10+1).3+(14+1).5=}{137}$	$\frac{(86+1).4+(98+1).5=}{843}$	0
Q_score ($\mu = 20$)		0	$20.(\frac{4}{0+1})$ =80	0	$20.(\frac{1}{3+1}+\frac{3}{21+1})$ =7.7	$20.(\frac{2}{1+1})$ =20	0	$20.(\frac{5}{70+1})$ =1.41
SSA_Z_score		15.72	6.20	20.12	-1.44	9.34	29.03	-1.19

which are indexed by search engines [22], to search for earlier questions and their answers that could be helpful to them. Tags are generic enough to convey semantic topics and, yet, specific enough to relate to programming concepts and expertise needed to fix *GitHub* bugs. As a sanity check against the possibility that tags may drastically limit the relevant information between *GitHub* and *Stack Overflow*, we examined the bug reports in three selected *GitHub* projects (out of the 20 projects considered in this study) and found that the textual information of each bug report (including projectLanguage, projectDescription, issueTitle and issueBody) mentions between 2 to 89 *Stack Overflow* tags (avg=14.9, var=132 and σ=11.5). In effect, the *Stack Overflow* tags define a common vocabulary for developers to exchange information. This vocabulary has a fundamental advantage over natural languages; all tags are useful and there is no need for stop-word and noise-word removal from the bug-report texts.

Our approach limits the search for potential bug assignees to the *Stack Overflow* members that have asked questions or provided answers with at least one tag in common with the text of the bug report under examination, *b*. To that end, we define the following terms.

$$A\_score_{u,b} = \sum_{a \,\in\, uanswers} (upVa + 1) \cdot (match\_tags_{a,b}) \quad (1)$$

$$Q\_score_{u,b} = \mu \cdot \sum_{q \ \in \ uquestions} \frac{(match\_tags_{q,b})}{(upVq + 1)} \quad (2)$$

$$Z\_score_u = \frac{(a - q)}{\sqrt{(a + q)}} \quad (3)$$

$$SSA\_Z\_score_{u,b} = \frac{(A\_score_{u,b} - Q\_score_{u,b})}{\sqrt{(A\_score_{u,b} + Q\_score_{u,b})}} \quad (4)$$

– **match_tags**$_{SO,b}$: all the *Stack Overflow* tags that appear in the title and description of the bug report; these are, in effect, the *Stack Overflow* topics that are important for the bug report in hand.
– **match_tags**$_{q,b}$: the shared tags between a question (q) and b.
– **match_tags**$_{a,b}$: the tags that annotate the question of an answer (a) that also appear in b.

Based on the above definitions, we have developed a measure of the expertise of the project developers in the areas defined by the $match\_tags_{SO,b}$ set. As we have discussed above, our inspection of numerous bug reports has established that the textual information of each bug report is usually matched with several tags. As a result, the relevant subsets of q and a for each developer, $match\_tags_{q,b}$ and $match\_tags_{a,b}$, frequently contain more than one elements.

Our expertise metric is specific to a particular bug report, b. It is *subject-aware* in that it considers two sets of tags –$match\_tags_{q,b}$ and $match\_tags_{a,b}$– relevant to the bug under examination. Finally, it is *social* in that it relies on social assessments of the *Stack Overflow* content, taking into account the numbers of upVotes and downVotes associated with the developer's *Stack Overflow* questions and answers.

Let us now describe our expertise metric for user u on bug report b. We define the $A\_score_{u,b}$ (see Equation 1) and $Q\_score_{u,b}$ (see Equation 2) to replace a and q respectively in the original definition of the Z_score. At any point in time, for every answer the user has contributed in the past that is relevant to the bug in question (*i.e.,* is associated with a tag that appears in the bug report), the number of $match\_tags_{a,b}$ is multiplied with the number of the answer's upVotes (plus one, for the answer itself). In effect, each answer contributes to the calculation of the user's expertise, taking into account the number of upVotes that the answer has received, which reflects the community's judgement on the answer's quality and usefulness. The sum of these terms make up $A\_score_{u,b}$. Each question is considered as evidence of lack of relevant expertise but this weakness is compensated by promotion of the question by other users (upVotes). To reflect the intuition that the "asker of a naive question is less knowledgeable than asker of a good one", we divide $match\_tags_{a,b}$ by the number of upVotes (plus one for the question itself). This tends to make the value of $Q\_score_{u,b}$ very small relative to $A\_score_{u,b}$, which is why we use the μ *normalization factor* to adjust it. The *social subject-aware Z_score* (*SSA_Z_score*) can then be defined as shown in Equation 4. This formula involves the terms relevant to the user's expertise (as $match\_tags_{q,b}$ and $match\_tags_{a,b}$ used in $A\_score_{u,b}$ and $Q\_score_{u,b}$ for different questions and answers). Furthermore, it takes into account the votes of the users to the answers and questions to advance good ones. The $SSA\_Z\_score_{u,b}$ focuses on answers and questions related to the topics relevant to the bug under examination.

Table 1 shows different scores for the users. Each cell at the intersection of a question and a developer contains the number of upVotes for the answer posted by that developer to the question. *Tom* has the best $SSA\_Z\_score_{u,b}$: he provided two answers to questions relevant to the bug, which received many upVotes.

Note that our implementation of the above score is aware of the temporal aspect of a developer's expertise. The activity of a developer in *Stack Overflow* accumulates over time but the estimation of the developer's expertise for a given bug report, reported in time t, is based only on his contributions up to date: the $SSA\_Z\_score_{u,b}$ considers questions and answers of the user u posted in time $t1 < t$.

A Recency-Aware SSA_Z_score The expertise of the developers shifts over time as they work on different projects with potentially different technologies [15]. Developers actively working in a particular domain are more appropriate to be assigned to a bug in this domain. This is why Mayter *et al.* consider a decay factor in their model of developers' expertise. Shokripour *et al.* [26] also consider this idea in their bug-assignment method: the older the evidence for a particular expertise is, the less relevant it is for current expertise needs. Anvik *et al.* [2] used filtering approaches to capture the recency of work.

Motivated by the intuition that "more recent evidence of expertise is more relevant", we define the *recency-aware, social, subject-aware* $RA\_SSA\_Z\_score_{u,b}$ as follows.

$$RA\_SSA\_Z\_score_{u,b} = \alpha \cdot (SSA\_Z\_score_{u,b}) + \beta \cdot \Big(\sum_{\substack{i\,\in\,previous\ bugs \\ assigned\ to\ u}} \frac{1}{1 + \substack{number\ of\ bugs\ occurred \\ between\ i\ and\ b}} \Big) \tag{5}$$

In this formula, α and β are tuning parameters and we explain how we tuned them in Section 4.4. Having the $RA\_SSA\_Z\_score_{u,b}$ for all users in the community over a bug report, our algorithm sorts the users and reports the top k as the most capable developers to fix the bug.

4 Evaluation

We obtained two *Stack Overflow* data sets [20][21] (approximately 65GB and 90GB). They consist of several XML files including information of 2,332,403 and 3,080,577 users, their posts, tags, votes, etc. In order to link these users to *GitHub*, their emailHash is needed [24][29], which is provided by the older data set. We merged these two data sets to get a large data set including the newer posts with old users.

We used a mySQL dump [23] (with a size of about 21GB) containing information of 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 provided by the same web site [23] (with a size of about 210GB) including information of 2,908,292 users. Again, we also merged the two data sets and obtained a large data set including information about *GitHub* users, projects and bug reports.

As our method assigns bugs to developers with a presence in both *GitHub* and *Stack Overflow*, we used identity merging [24][29] to identity the common users in *GitHub* and *Stack Overflow.* The *GitHub* data set contains the e-mails of the users, but *Stack Overflow* data set includes e-mail hash. So for each *GitHub* user, using MD-5 function, we obtained the e-mail hash and compared it with e-mail hashes in *Stack Overflow.* With this approach, we found 358,472 common users.

4.1 Experiment Setup

For each *GitHub* project, we first calculated the union of the sets of project members, committers, bug reporters and bug assignees, and we removed from this set all developers without any *Stack Overflow* activity, to calculate the project's community-members set. Next, we sorted the projects based on the cardinality of their community-member sets and we identified the top 20 projects[3] with the highest number of community members and the highest number of bug assignees.

For the selected 20 projects, the number of community members vary from 28 to 822 (average=127, 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 project's community members. Note that we could not use the rest of bug reports since they were assigned to developers with no *Stack Overflow* activity. 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 of the 20 chosen projects, we ran our algorithm to recognize the $\text{RA\_SSA\_Z\_score}_{u,b}$ score of all project-community members. Then, we ranked the users from the highest score to the lowest.

We report the average *top-k* recommendation accuracies. We compare our results for k=1 and k=5 with several implemented methods, as well as previously published results. We also report our results based on MAP (Mean Average Precision) as a precise, synthesized, rank-based evaluation measure.

4.2 Comparison to State of the Art

Direct comparison with earlier methods is not possible since none of the previous studies we reviewed above have made available their bug-assignment algorithm implementation and data sets. To approximate this comparison, we experimented with the `scikit-learn` [4] implementations of a number of algorithms classify-

[3] rails/rails, scala/scala, adobe/brackets, JuliaLang/julia, mozilla/rust, mozilla-b2g/gaia, angular/angular.js, bundler/bundler, lift/framework, dotcloud/docker, edx/edx-platform, elasticsearch/elasticsearch, fog/fog, html5rocks/www.html5rocks.com, Khan/khan-exercises, saltstack/salt, travis-ci/travis-ci, NServiceBus/NServiceBus, TryGhost/Ghost and yui/yui3

[4] `http://scikit-learn.org/stable/`

ing bugs to developers, which we applied to our own data set. Considering the previous bug reports and the real assignee for each one, these algorithms use word-based features of the bug reports to predict the most probable developer who would fix the bug.

1NN, 3NN and 5NN In this family of classifier methods, each bug report is considered a point in a multi-dimensional space, each dimension defined by a distinct word. Each developer (class) corresponds to a hyper-plane in this space, consisting of all the bugs closed by the developer. Then, given a new bug report and a corresponding new point in the space, the closest existing point is selected. The class of the selected point (bug report) is the recommendation for the new bug report. This process is called Nearest Neighbor (1NN). In 3NN and 5NN, we look for 3 or 5 nearest points (bug reports) to that point and simply get their average to determine a hyper-plane and its class (developer) as the recommendation. Lamkanfi, *et al.* [12] and Anvik [3] used this method for their predictions about bug reports.

Naive Bayes (NB) and Multinomial Naive Bayes (MNB) In this family of algorithms, the developers' features are the words included in the textual elements of the bug reports they have handled before. These features are considered by the learner as a bag of words. Given a new bug report, the classifier returns the classes (developers) with the highest number features in common with the bug. Bhatacharya *et al.* [5], Čubranić and Murphy [8] and Anvik [2] are from those researchers who used this method for bug triaging.

Building on the above method, a group of Naive Bayes classifiers, one per developer, may be constructed to decide the developer to which a given bug report belongs, and to calculate the probability of that being the case. Then, this probability is compared over all the developers to infer the most probable bug fixers. Lamkanfi *et al.* [12] and Anvik [3] used this method for bug triaging.

SVM This approach represents bug reports as vectors in a multi-dimensional space –similar to 1NN, 3NN and 5NN. With each word being a dimension, this classifier considers each bug report a point in this multidimensional space. Then, considering all the bug reports that are already assigned to each developer as a *category*, the optimal hyper-planes between these points to separate different categories is inferred. This method also assigns a label (name of a developer) to each category. Then, given a new bug report, it reports the label of its category. Lin *et al.* [13], Anvik *et al.* [2] and Bhattacharya *et al.* [5] used this method for bug triaging.

4.3 Implementation

The Java implementation of our approach as well as our data sets (3 training and tuning and 17 final evaluation projects and their bug reports) and output results are available online at `https://github.com/anonymous-user-1/BugTriaging` for consideration or future comparisons.

Regarding the implemented Machine-Learning approaches, given that no open implementations were available for the previous bug-assignment methods reported in the literature, we made fair effort toward the best implementation of the competitor algorithms. We processed bug reports' title and body words with

TFIDF, producing TFIDF word vectors. In order to make the process competitive enough to our approach, we made the process online; train them on first n-1 bug reports and then test on the n$^{\text{th}}$. Then train on first n bug reports and test on n+1$^{\text{th}}$ and so on.

We used the followings 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) class. We chose RBF kernel type, used shrinking heuristic, with gamma kernel coefficient $1/n$ for n features, error penalty=1 and probability=true. More details as well as the the Python implementation of the mentioned approaches are available online at `https://github.com/anonymous-user-1/ML-bug-triager-scikit/blob/master/dumpbayes.py`.

4.4 Performance of Variant Social Metrics of Expertise

In Section 3 we incrementally developed our *Triage_score* starting with the simple social measures of expertise a and q. To gain an insight on how each aspect of this measure contributes to the bug-assignment effectiveness, we applied several intermediate variants of the metric, representing different intuitions in its evolutionary construction process, to three test projects with 490 bug reports in total, randomly selected from the 20 projects of our study.

The performance of the simplest measure, *i.e.,* the number of answers, *AnswerNum* [31], tagged with at least one of those $match\_tags_{SO,b}$ is shown in Table 2. The triaging accuracy is poor and does not recommend this naive measure for the bug-assignment task.

The original Z_score [31], which considers answers as indication of expertise and questions as indication of lack of expertise, does not perform much better. The problem was that the Z_score metric measures general expertise rather than expertise specific to the bug under examination, and, as a result, it is inadequate to compete with the approaches reported in the literature.

Next we evaluated the subject-aware Z_score, *SA_Z_score*, which measures expertise of the developers in $match\_tags_{SO,b}$, without considering upVotes. This score is in effect equivalent to *SSA_Z_score*, but with μ=1 and without considering upVotes. μ was the *normalization factor* which we used to balance the values of $Q\_score_{u,b}$ with $A\_score_{u,b}$ when it was divided by "1+number of upVotes of the question". In other words, we set $\mu = 1$ for *SA_Z_score* because it does not consider upVotes. Again, a small improvement was observed in the performance, evidence that, not surprisingly, awareness of the bug under examination is useful in selecting the right bug assignee. Still this score is not competitive with the literature results.

Our next step was to consider the community's curation of the questions and answers. Instead of uniformly considering all *Stack Overflow* answers of a developer as evidence of expertise and all questions as evidence of lack of expertise, we evaluated whether weighing "good" answers and questions more than "bad" ones would make a difference. The *Stack Overflow* users' upVotes

Table 2. Accuracy results for preliminary approaches and tuning

Method		Top-1	Top-5	MAP
AnswerNum		3.40	21.00	0.1384
Z_score		3.49	21.05	0.1453
SA_Z_score (μ=1, upVotes=0)		9.12	23.59	0.1801
SSA_Z_score	μ=1	**12.33**	56.97	0.3216
	μ=10	12.06	52.68	0.3153
	μ=20	11.79	50.67	0.3128
	μ=1+avg(upVotes)	12.06	53.61	0.3166
	μ=1+avg(upVotes)^2	11.66	53.73	0.3130
	μ=1+HM(upVotes)	**12.33**	**58.45**	**0.3223**
recency-aware SSA_Z_score	α=0.001	42.65	88.37	0.618
	α=0.01	**43.06**	**88.57**	**0.621**
	α=0.1	41.84	86.33	0.609
	α=1	39.59	77.96	0.565
	α=10	38.98	77.14	0.559

are evidence for the quality of the questions and answers and the *social subject-aware Z_score* (*SSA_Z_score*) was designed to take them into account, as well as being aware of the bug context. This metric involves the μ *normalization factor* that determines the importance of considering "asking" as "lack of expertise" with respect to answers. It can be assigned a static value, or, it may be tuned for different projects. For all projects, we set it to *"1+Harmonic Mean of upVotes of all related questions"* (all questions containing at least one $match\_tags_{SO,b}$) which has slightly better performance. The tuning results are shown in Table 2. Note that for the example of Table 1, we have μ=20, obtained simply based on the average of upVotes of the questions mentioned in the first column.

The final improvement leading to our triage score was to make it sensitive to the recency of the relevant *Stack Overflow* activity. The key intuition here is that "the fixing activity has locality" meaning that "the recent fixing developers are likely to fix bug reports in the near future" [27]. Inspired by this idea, we considered the recency of the developers" activities, highlighting recent ones more than past ones. As we anticipated, the results improved further.

Finally, we examined the impact of the various parameters of our metrics to the bug-triaging performance. For the purpose of tuning and calibrating our method, we needed to determine the values for α and β in the $RA\_SSA\_Z\_score_{u,b}$ (Equation 5). We set the value of β to 1 in order to reduce the variables to one. Then, changed α and measured the accuracy and MAP on three test projects. The best results obtained with α=0.01. This is 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). Later in this section, we apply the parameter values (μ, α and β) obtained from the three projects into the remaining 17 projects in our final evaluation.

4.5 Performance of the $RA\_SSA\_Z\_score_{u,b}$

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. We measured the average *top-k* accuracies as well as MAP. 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 the MAP as 0.633, which is very strong and shows that the harmonic mean of the real assignee is 1.58 over all the bug reports.

We also implemented the other approaches discussed in Section 4.2. 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 3.

Table 3. Accuracy results for different simulated approaches compared with ours

	1NN	3NN	5NN	Naive Bayes	Multinomial Naive Bayes	SVM	Our Approach
Top1 Accuracy (%)	43.09	**46.48**	45.60	43.77	42.75	45.46	45.17
Top5 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**

Note that all the values reported in Table 3 are averages over all the 17 projects examined; due to paper-length limitations, the per-project values are not shown here. However, we examined the detailed results for each project and found them close to the mean (var=60.97 and σ=7.81 for *top-5* accuracies). Our results demonstrate that our $RA\_SSA\_Z\_score_{u,b}$, 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. In the next section, we analyze these results and compare the details with the other methods.

5 Analysis

First, we compare our approach against implemented machine-learning methods. The results in Table 3 show that our method outperforms all of the 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 ours). Comparing the different algorithms on the same data set demonstrates the usefulness of our method. The improved MAP and accuracy of our approach over these other methods shows that our approach is trustworthy and capable of precise assignee recommendation.

Let us now compare the accuracy of our approach against the accuracy reported in previous published contributions. Due to differences in the experimental design and collected metrics of the various studies, it is impossible to have an exact and fair comparison. Some of these earlier methods reported the maximum accuracy over different projects instead of the average accuracy. Also they differ in reported values for k in *top-k* accuracies, with *top-1* and especially *top-5* being the most frequently used. As one of the best obtained accuracies in the previous studies, Shokripour *et al.* obtained 48% *top-1* and 60% and 89% *top-5* accuracies on two projects (between 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. In fact, some of the features and meta-data that are required for their method (*e.g.,* product and component of the bug reports) are quite difficult to obtain [12], which makes this study quite challenging to eplicate.

To summarize our comparison findings, it is important to mention the following. 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.

Limitations and Threats to Validity The most important concern with respect to the validity of our method is that the common users (between *Stack Overflow* and *GitHub*) who constitute the project community are a small part of the complete set of developers associated with each project. The common users between *Stack Overflow* and *GitHub* represent up to 20% of the total number of users, in each of these networks. There are many users about whom we do not have information, because we could not match their profile in the two networks. However, to mitigate this limitation, unlike most previous studies, we examined our approach on a large number (*i.e.,* 20) of big projects with thousands of users and bug reports which is quite substantial, limiting threats to external validity. We can even argue that this phenomenon may be an advantage of our approach that focuses on high-quality evidence of developers' expertise established in the actively curated *Stack Overflow* community and ignores developers who do not have such credentials. If our method performs well by accessing parts of the developers' contributions, it should improve when accessing the complete information.

Currently, for privacy reasons, much of the Q&A content at the software social networks is provided anonymously. One could envision however that project managers could request their developers to provide their *Stack Overflow* IDs. Thus, the step of identifying users common across the two networks through their e-mails should become unnecessary and a larger community of developers, with far more extensive Q&A contributions, will become available to the bug-assignment process.

One concern, is the practice of some developers answering their own questions on *Stack Overflow* for announcing a commonly encountered issue with some API, library, etc. However, we investigated the questions and answers of members of three (out of the 20) chosen projects and found that only 3% of their answers are answers to one's own questions, and only in around half of these cases the question is up-voted, meaning that the case did not indicate expertise, but lack of expertise (as we assumed).

6 Conclusions and Future Work

The fundamental novelty of our work lies in that it is the first bug-assignment method to consider evidence of developers' expertise beyond their contributions to software development, examining instead their contributions to a Q&A platform. Our method takes advantage of the fact that many developers participate in both platforms. Relying on the expertise of the community to recognize good (and bad) questions and answers, our method taps into a rich, and as yet unexploited, social source of expertise information. To consider this information in the context of the software-development task at hand, our method relies on the intersection between *GitHub* bug-report text and tags of the *Stack Overflow* questions and answers. We believe *Stack Overflow* is a rich source of expertise for software engineering purposes since the privilege of important *Stack Overflow* contributions like up/downVoting is only available to community members who have established a minimum reputation.

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 exploits expertise information found in *Stack Overflow* and readily outperforms the competition. We believe that in order to achieve even better performance, a project manager may ask the ID of his developers in the software social networks and identify their full Q&A contributions.

Generalizing beyond *Stack Overflow*, how helpful it is for bug assignment, and what limitations it suffers, we envision a new research agenda studying the application of third-party expertise networks to bug triaging. The biggest open question is how to generalize this approach to multiple expertise networks. As well as various Q&A networks and code forums, perhaps there are wikis, project documentation, or developer performance histories that could be mined for expertise networks to exploit for bug triage.

In addition to considering multiple social platforms, we also plan to consider tag synonyms: *Stack Overflow* introduces lists of tag synonyms and suggests the users to use the primary definitions (e.g., "servlets" instead of "webservlet", "authentication" instead of "login"), but does not enforce the practice. In the future, we plan to consider integration of the synonyms in their primary definitions in code and data sets.

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. Anvik, J.: Automating bug report assignment. In: Proceedings of the 28th International Conference on Software Engineering (Doctoral Symposium). pp. 937–940. ICSE '06, ACM (2006)
2. Anvik, J., Hiew, L., Murphy, G.C.: Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering. pp. 361–370. ICSE '06, ACM (2006)
3. Anvik, J.K.: Assisting Bug Report Triage through Recommendation. Ph.D. thesis, University of British Columbia (Nov 2007)
4. Baysal, O., Holmes, R., Godfrey, M.W.: Revisiting bug triage and resolution practices. In: In Proceedings of the User evaluation for Software Engineering Researchers (USER) Workshop at the International Conference on Software Engineering (ICSE), 2012. pp. 29–30. IEEE (2012)
5. Bhattacharya, P., Neamtiu, I., Shelton, C.R.: Automated, highly-accurate, bug assignment using machine learning and tossing graphs. Journal of Systems and Software 85(10), 2275–2292 (2012)
6. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems 30(1), 107–117 (1998)
7. Canfora, G., Cerulo, L.: Supporting change request assignment in open source development. In: Proceedings of the 2006 ACM Symposium on Applied Computing. pp. 1767–1772. SAC '06, ACM (2006)
8. Čubranić, D., Murphy, G.C.: Automatic bug triage using text categorization. In: In SEKE 2004: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering. Citeseer (2004)
9. Fritz, T., Ou, J., Murphy, G.C., Murphy-Hill, E.: A degree-of-knowledge model to capture source code familiarity. In: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1. pp. 385–394. ICSE '10, ACM (2010)
10. Jeong, G., Kim, S., Zimmermann, T.: 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. pp. 111–120. ESEC/FSE '09, ACM (2009)
11. Kleinberg, J.M.: Hubs, authorities, and communities. ACM Computing Surveys (CSUR) 31(4es), 5 (1999)
12. Lamkanfi, A., Demeyer, S., Soetens, Q.D., Verdonck, T.: Comparing mining algorithms for predicting the severity of a reported bug. In: Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. pp. 249–258. IEEE (2011)
13. Lin, Z., Shu, F., Yang, Y., Hu, C., Wang, Q.: 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. pp. 451–455. ESEM '09, IEEE Computer Society (2009)
14. Linares-Vásquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M., Poshyvanyk, D.: Triaging incoming change requests: Bug or commit history, or code authorship? In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on. pp. 451–460. IEEE (2012)
15. Matter, D., Kuhn, A., Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers. In: Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on. pp. 131–140 (May 2009)
16. Mockus, A., Herbsleb, J.D.: Expertise browser: A quantitative approach to identifying expertise. In: Proceedings of the 24th International Conference on Software Engineering. pp. 503–512. ICSE '02, ACM (2002)

17. Naguib, H., Narayan, N., Brugge, B., Helal, D.: Bug report assignee recommendation using activity profiles. In: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE (2013)
18. Nguyen, T.T., Nguyen, A.T., Nguyen, T.N.: Topic-based, time-aware bug assignment. SIGSOFT Softw. Eng. Notes 39(1), 1–4 (Feb 2014)
19. Poshyvanyk, D., Marcus, A.: Combining formal concept analysis with information retrieval for concept location in source code. In: Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on. pp. 37–48. IEEE (2007)
20. 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
21. Stack Exchange, Inc: Stack exchange data dump, "https://archive.org/details/stackexchange", Visited on 2014/08/20
22. Stack Exchange Team: What are tags, and how should i use them?, "http://stackoverflow.com/help/tagging", Visited on 2015/03/17
23. The GHTorrent Project: Mysql database dumps, "http://GHTorrent.org/downloads/mysql-2014-08-18.sql.gz", Visited on 2014/08/20
24. Sajedi, A., Esteki, A., GholiPour, A., Hindle, A., Stroulia, E.: Involvement, contribution and influence in github and stack overflow. In: Proceedings of the 2014 Conference of the Center for Advanced Studies on Collaborative Research. CASCON '14, ACM, Markham, Toronto, Canada (2014)
25. Shokripour, R., Kasirun, Z., Zamani, S., Anvik, J.: Automatic bug assignment using information extraction methods. In: Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on. pp. 144–149 (Nov 2012)
26. Shokripour, R., Anvik, J., Kasirun, Z.M., Zamani, S.: 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. pp. 2–11. MSR '13, IEEE Press (2013)
27. Tamrawi, A., Nguyen, T.T., Al-Kofahi, J., Nguyen, T.N.: Fuzzy set-based automatic bug triaging (nier track). In: Proceedings of the 33rd International Conference on Software Engineering. pp. 884–887. ICSE '11, ACM (2011)
28. Tamrawi, A., Nguyen, T.T., Al-Kofahi, J.M., Nguyen, T.N.: Fuzzy set and cache-based approach for bug triaging. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. pp. 365–375. ESEC/FSE '11, ACM (2011)
29. Vasilescu, B., Filkov, V., Serebrenik, A.: Stackoverflow and github: associations between software development and crowdsourced knowledge. In: Social Computing (SocialCom), 2013 International Conference on. pp. 188–195. IEEE (2013)
30. Venkataramani, R., Gupta, A., Asadullah, A., Muddu, B., Bhat, V.: Discovery of technical expertise from open source code repositories. In: Proceedings of the 22Nd International Conference on World Wide Web Companion. pp. 97–98. WWW '13 Companion, International World Wide Web Conferences Steering Committee (2013)
31. Zhang, J., Ackerman, M.S., Adamic, L.: Expertise networks in online communities: Structure and algorithms. In: Proceedings of the 16th International Conference on World Wide Web. pp. 221–230. WWW '07, ACM (2007)
32. Zimmermann, T., Nagappan, N., Guo, P.J., Murphy, B.: Characterizing and predicting which bugs get reopened. In: 34th International Conference on Software Engineering (ICSE), 2012. pp. 1074–1083. IEEE (2012)