Realistic Bug Triaging

Ali Sajedi Badashian
Department of Computing Science, University of Alberta, Edmonton, Canada
alisajedi@ualberta.ca
http://webdocs.cs.ualberta.ca/~sajediba
Advisor: Professor Eleni Stroulia
stroulia@ualberta.ca

ABSTRACT

Bug triaging, i.e., assigning a bug report to the developer "best" able to address it, involves identifying a list of developers qualified to understand and address the bug report and ranking them according to their expertise. Most research in this area addresses this task by matching the description of the bug report and the developers' prior development and bug-fixing activities.

This thesis puts forward a more realistic formulation of the bug-triaging task. First, we develop a novel model of the developers' expertise, taking into account relevant evidence from their code-development conributions, as well as their contributions to relevant Question-and-Answer (Q&A) platforms. Second, we adopt an economics perspective to the task, and we propose to generalize bug-triaging from "assigning one bug to the best developer" to "cost-effectively assigning multiple peding bugs to a set of qualified and available developers." In this paper, we report on our early results on the value of broadening the notion of developer's expertise to take into account evidence from Q&Aplatforms.

1. INTRODUCTION AND BACKGROUND

The task of bug triaging, i.e., assigning a bug report to the "best" person to address it, is typically formulated as ranking the available developers according to their suitability to fix a given bug report. Most research in this area matches the bug report with developers, by examining the description of the bug report and the developers' prior development and bug-resolving activities, and selects the *top-k* most capable developer(s) to resolve the bug in question.

Table 1 summarizes the key publications in this area and their results. They all share the same notion of "expertise", considering only the modules that developers have "touched" and the bugs they have fixed. Surprisingly however, even though some publications claim bug-assignment with extremeley high accuracy, up to 80% [9, 3], in practice, around 40% of the bugs are tossed between developers [5]. We are hypothesizing that this discrepancy is due to the rather simplistic approach of current research to the bug-triaging problem. This is why, in this thesis, we put forward a more realistic formulation of the problem, considering (a) a broader evidence of the developers' expertise, and (b) the economic trade-offs of assigning bugs to competent developers while at the same time taking into account their workload, availability and cost. Our notion of expertise includes evidence from their contributions to code development, as well as their contributions to social Q&A platforms. Further, we generalize the specification of the bug-triaging task, from "assigning one bug to the developer most able to fix it" to "cost-effectively assigning multiple peding bugs to a set of qualified and available developers."

With this objective, this thesis will first model the developers' expertise, inspired by expert finding approaches in social networks [2] considering their work record (including their code-development, documentation, and bug-fixing contributions as well as their answers on Q&A platforms), the community's opinion of this record (as evidenced be bug refixes, and answer ratings), and their status in the community (based on their project roles and their seniority in Q&A platforms). We plan to evaluate our work in the context of GitHub (the main code sharing website) and Stack Overflow (the most popular Q&A website), in three steps:

1. In order to validate the relevance of Q&A contributions as evidence of a developer's expertise, we developed a method that, given a bug, identifies candidate assignees based on the common terms between the bug description and the developer's questions and answers in Stack Overflow, and the Stack Overflow community's rankings of these consributions. Intutively, if a developer has answered several questions on keywords relevant to the bug under consideration, and her answers have received the community's approval with many upVotes, then she has a proven expertise record relevant to the bug and she should be a likely candidate for fixing it[14].

2. Next, we will extend this model with more evidence of the developer's work record. To date, researchers have considerd code-development and bug-fixing; in addition, we plan to consider documentation and testing activities. Intuitively, if a developer has documented a piece of code, she is likely competent to contribute to its further development and fixing. Similarly, if she has developed unit tests for a class, she is likely an expert on this class also. In general, we plan to analyze all these types of contributions to a project to etxract a set of terms with which to characterize the developer's expertise.

3. Finally, we plan to refine the current methods of keyword-matching between the developer's profile and the bug description with a graph model for detecting the specificity and relatedness of the topics (i.e., keywords). In this model, we will build a weighted graph of connected topics and we will use its sub-graphs as a developer's set of expertise and the topics of interest in a bug report.

Then, in the second part of the thesis, we will address the problem of cost-effectively assigning multiple bugs to multiple developers. The intuition is that focusing on one bug at a time may result in overloading the few, most capable developers with too many bug assignments. Taking into account real-world constraints, such as developers' salaries and workloads, we will formulate the bug-assignment problem as

Table 1: Recent Bug-Triaging Methods

Authors	Basic method / Information used	Best accuracy
Čubranić and Murphy[4]	Naive Bayes classification of bug reports (i.e., "text documents") to developers (i.e., "classes"); uses bug summary and description.	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.	64% *top-3*
Tamrawi et al. [18]	A fuzzy-set representation of the relations between developers and the bug-reports' technical terms; uses bug summary and description.	average 40% *top-1* and 75% *top-5*
Lamkanfi et al. [9]	Multinomial Naive Bayes for predicting severity of bug reports; uses bug-report summary, description, severity and component.	79%
Lin et al. [10]	SVM and C4.5 classifiers; uses bug summary and description, type, class, priority, submitter and the module ID.	77% *top-10*
Canfora and Cerulo [3]	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.	85%
Matter et al. [13]	Vector Space Model (an IR method); uses source-code commits and the bug-report keywords;	34% and 71% *top-1* and *top-10*
Linares-Vásquez et al.[11]	IR-based concept location techniques; uses text of a change request and source code files.	85% *top-5*
Shokripour et al. [17]	A location-based approach (using information extraction); uses bug summary and description, detailed source code info (comments, names of classes, methods, fields, etc.).	48% *top-1* and 89% *top-5*
Jeong et al. [5]	"Tossing graphs" of developers + ML approaches; uses bug report title and description	77% *top-5*

a "job-assignment problem" [6], and we will devise solutions based on the "Hungarian algorithm" [8]. These solutions will optimize the overall relevant expertise of the developers to their assigned bugs, while balance the developers' workload, and minimize the overall cost of the task (i.e., the developers' salaries). We argue that this new formulation of the bug-triaging, under the more holistic notion of develoeprs' expertise, is realistic and much more useful in practice.

2. RESEARCH PROBLEM AND METHODOLOGY

In this section, we discuss the two aspects of this thesis, as well as our plans to evaluate our methods.

2.1 A Holistic Evidence of Expertise

As we mentioned earlier, in order to capture the users' expertise, we consider three types of evidence; the users' work record (code documentation, bug fixing or question answering), opinions about them (the ratings their answers attracted) and their community status (project membership). We implement two models to capture expertise using these evidences. Then, revise its topic matching in the third one.

2.1.1 Bug triaging using Q&A contributions

In our first model we cross-referenced GitHub and Stack Overflow for bug triaging and used Stack Overflow contributions in identifying expertise and assigning bugs to the developers [15]. We developed SSA_Z-score which is a social and subject-aware metric that captures expertise of the developers in specific areas (mentioned in the bug report) and takes into account social contributions of the users (i.e., votes) as well. In this model, we used Stack Overflow tags for cross-referencing between Stack Overflow and GitHub contents and showed that this approach outperforms other methods.

2.1.2 Inclusion of previous bug fixing history

We propose to take into effect the users' development activities (which are not fully utilized) as well as their social contributions. So we identify all the tasks (comments, bug reports, code documentations) handled by them and extract their text. Then, we assign a document to each developer including the text from all his tasks up to that point cumulatively (this needs an agile indexing scheme). Then –using an Information Retrieval technique– when a new bug report arrives, we use its text as a query and calculate the TF-IDF (term frequency-inverse document frequency) [12] measure over all available documents (developers) and give each of them a score. Finally, we will combine this score with SSA_Z-score in our current triaging approach.

2.1.3 Expertise Graph Model

Currently, in order to match a bug report with a question/answer, we look for all shared tags using a simple tag matching. However, there are two possible concerns:

- **Specificity, relevance and importance**: Some tags are general keywords and some are detailed ones. Usually, when it comes to describe an issue, problem or bug report, the specific terms are much better than the general keywords.
- **Connectivity and relationships**: Only a few of the needed topics are usually mentioned directly in the bug report. However, topics (i.e., tags) are connected to each others. These connections can emphasize on a specific tag, or even lead us to a new one that is not currently in the list of needed tags.

Usually, having a bug report at hand, the needed expertise is a set of topics (i.e., tags) not just a single topic. In our previous study [15], We observed that in average there are almost 15 Stack Overflow tags mentioned in a bug report. These are the set of keywords that we are looking for experts in them. This set is a subset of an overall keyword model that we use in order to build an expertise tag model with capability of highlighting important keywords as well as inferring new keywords to better coordinate the GitHub bug reports with Stack Overflow questions and answers.

In this model, we will build a weighted graph of all Stack Overflow tags connected to each other. The nodes represent tags and the edges represent the relationships between them. Both nodes and edges have weights (shown respectively with node size and edge thickness) as follows:

Node size represents specificity (unlike general reputation or usage) so that the more specific the keyword is, the bigger the node will be. On the other hand, the more general keywords are shown smaller. For example, "search" which is a general keyword is smaller than "elasticsearch" and both are smaller than "elasticsearch-plugin" which is a specific one. The rationale behind this distinction of the nodes is their *specificity*: we want to focus on specific terms that are in the bug report than the generic keywords. For example, suppose "elasticsearch-plugin" and "java" are mentioned in a bug report. However, the former is of more importance than the later because due to broadness of "java", many people know about (some parts of) it, but only a small fraction of them know about "elasticsearch-plugin". By making the specific nodes bigger, the spotlight will be on exact, detailed requirements rather than some generic terms. Obtaining the node sizes will be based on the (inverse of) number of occurrence of them in different questions.

Thickness of the edges represent the connectivity/relationships between connected tags. The thicker the edge is, the more relevant the two connected tags are. For example, the edge between "ajax" and "jquery" is thicker than the edge between "ajax" and "c#". There is a high chance that a developer did not answer a question about a requested topic in hand, but answered regarding another highly related topic and could be a good candidate regarding that original topic in hand. For example, the users who answered about "ajax" are usually knowledgeable about "jquery" and should be considered as a candidate. Similarly, each question in Stack Overflow is tagged with only a few related tags and there is a high possibility that some important ones are missed. As a result, considering the thickness of the edges in the graph of tags, we will make inferences regarding the developers' expertise in new (or less-participated) topics as well as topics of questions/answers that are not determined originally. The thickness of an edge will be obtained based on the number of co-occurrences of its two connected tags in all questions.

After building this general tag model, we will locate three important types of sub-graphs; the graph of expertise of a developer, the graph of areas of a bug report (required expertise areas) and the graph of areas of a question/answer. As a result, we will change the simple tag matching to expertise graph matching. Needless to say, the result of each matching will no longer be 0 or 1, but a fuzzy score. Having a bug report at hand, we look for the developer with closest expertise graph to that of the bug report. In the simple case, this can be obtained through either maximum similarity or minimum difference [7, 20] between the desired set and the developers' set. We finally propose methods to generalize this methodology from Stack Overflow tags to the concepts of code and documentation (and essentially any other source of keywords). We believe this will be a unique, novel contribution in both bug assignment and expertise identification.

2.2 Multiple Bug Assignment

To date, bug-assignment research has focused on developer selection, maximizing the developer's relevant expertise and her potential to work on a single bug report under consideration. This view ignores two important real-world constraints.

First, at any point in time, many different bugs are pending and several of them may be equally urgent; therefore, there is no reason why anyone of them should be chosen before all others. Consequently, we consider the problem of selecting developers to fix a total of B bug reports, given D available developers. For simplicity, we assume that $B \leq D$. Note that the general case (in which there are more bug reports than developers) can be decomposed into two or more such simple assignment problems.

Second, while expertise is a determining factor in selecting a developer to fix a bug, there are many other relevant economic and operational constraints that should not be neglected. We will consider minimizing cost (as the sum of the salaries of the developers working on different assigned bugs) and balancing availability (as a function of the current workload balancing of the project developers) as two important objectives of the bug-assignment process, in addition to maximizing the developers' expertise. Intuitively, assigning a bug to the most expert developer may cost unnecessarily more than an alternative developer who is slightly less expert in the field, but much more junior and therefore less expensive. Furthermore, there is little point in assigning several bug reports at a time to one expert developer, ignoring the consequent high workload of this person. In our work, we assume that each developer receives a predefined wage, which impacts the cost of the bug-resolution process. In addition, we also assume that it is impossible to assign more than a threshold number of bug reports to any single developer (so as not to overload the developers).

We will address this formulation of the bug-assignment problem incrementally, through the development of a sequence of increasingly complex models:

- **Single bug assignment**: Each developer has a defined wage, w_i, and can be assigned to 0/1 bug report. Thus, the task is to optimize **"the distribution of bug reports evenly between developers, in order to maximize average expertise and minimize the total wage"**. In this model, we will handle expertise and cost.
- **Multiple bug assignment**: Each developer has a defined wage, w_i. As an availability factor, any developer can be assigned up to k bug reports to handle. Moreover, for workload balancing purposes, for each assignment to a developer after the first one, the cost will be increased by a factor of f so that each assignment is charging $\%f.w_i$ additional cost compared to the previous one. In other words, the wage for the jth bug assignment to the same developer will be $w_i + \frac{(j-1).f}{100}.w_i$. We will replace k and f with some constants (e.g., $k = 3$ and $f = 10$), but will propose our models dynamically based on the factors k and f to be generic enough (applicable for any project). Hence, we have the optimization problem **"to assign the bug reports to developers in order to maximize average expertise and minimize the total wage while considering the threshold of k bug reports for each developer and preserve workload balancing"**.

In a novel mapping, we will convert these two problems to (variations of) "assignment problem" [6] which is a fundamental combinatorial optimization problem. This algorithm finds a matching with maximum (or minimum) resulting value in a weighted bipartite graph. Then, we will use the well-known Hungarian (Kuhn-Munkres) algorithm [8] and innovatively adapt it with the new shape and constraints of our problem to devise our multidimensional triaging algorithm. This is a unique adaptation of the assignment problem and Hungarian algorithm that satisfies the demands of our proposed multidimensional bug triaging.

2.3 Evaluation Plan

In order to evaluate our approaches, we will use the available sets of bug reports in 20 top GitHub projects as we did for evaluating our primary approach of Section 2.1 in [15]. In each of the remaining steps (Sections 2.1.2, 2.1.3 and 2.2), we will run our algorithm and rank the users for each available bug report. Then, we will cross-validate the recommendations of our approach with the real assignee (available in the data set) and measure the results in terms of *top-1* and *top-5* accuracies (as the mostly used measures [15]) as well as MRR (Mean Reciprocal Rank) [19]. Then, compare them with our currently available results (Section 2.1.1 and the publication results of our currently published ICSME paper [15]), which is already shown to outperform previous approaches.

In the last part (Section 2.2), however, we also consider overall cost, availability and load balancing. The real values for these factors are not available. So we will compare these three factors against our previous approach [15] and report the results on every factor separately. We can also compare all the four metrics against the greedy algorithm.

3. WORK PROGRESS, PRELIMINARY RESULTS AND PUBLICATIONS

We first obtained various data sets of GitHub and Stack Overflow. Then, since we needed the GitHub's data frequently, we implemented a tool as a data adaptor for GitHub's data [16]. We also performed a comprehensive analytical research on the interplay between the two networks and published the results [14]. Then, in our ICSME 2015 paper, "Crowdcourced Bug Triaging"[15], we showed that this methodology (Section 2.1.1 above) works well. There is another research currently under review (FASE 2016) regarding Section 2.1.1. We have also designed the algorithms and structures needed for the next steps.

4. ANTICIPATED CONTRIBUTIONS

The main contributions of this thesis will be as follows:

1) **A comprehensive notion of expertise based on social networks and graph theories**: Fusing different types of evidence from GitHub and Stack Overflow, We will develop a novel comprehensive expertise model that takes into account multiple aspects of work and the community's appreciation of this work.

2) **A graph based approach in modeling topics and their network**: We will build a weighted network of topics based on the question answering contributions in Stack Overflow (and we will introduce methods to generalize it to other development sources). This model is useful in identifying the most important technical topics of a corpus of text without being supervised by human.

3) **A realistic bug triaging model**: We will integrate economics trade-offs in software engineering practice of bug triaging while maximize bug-developer affinity.

5. ACKNOWLEDGMENT

This work was supported by Grand postgraduate scholar award (by Canadian Grand-NCE) and is being supported by Alberta Innovates-Technology Futures graduate student scholarship.

I would like to thank my advisor, professor Eleni Stroulia for her support.

References

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of International Conference on Software Engineering*, ICSE '06, pages 361–370. ACM, 2006.

[2] Krisztian Balog, Yi Fang, Maarten de Rijke, Pavel Serdyukov, and Luo Si. Expertise retrieval. *Foundations and Trends in Information Retrieval*, 6(2–3):127–256, 2012.

[3] Gerardo Canfora and Luigi Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1767–1772. ACM, 2006.

[4] Davor Čubranić and Gail C. Murphy. Automatic bug triage using text categorization. In *Software Engineering & Knowledge Engineering (SEKE)*. Citeseer, 2004.

[5] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. ESEC/FSE '09, pages 111–120. ACM, 2009.

[6] Bernhard Korte and Jens Vygen. *Combinatorial optimization: Theory and Algorithms*. Springer-Verlag, 2002.

[7] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. Technical report, Carnegie-Mellon-University, 2011.

[8] Harold Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[9] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. Comparing mining algorithms for predicting the severity of a reported bug. In *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*, pages 249–258. IEEE, 2011.

[10] Zhongpeng Lin, Fengdi Shu, Ye Yang, Chenyong Hu, and Qing Wang. An empirical study on bug assignment automation using chinese bug data. ESEM '09, pages 451–455. IEEE Computer Society, 2009.

[11] Mario Linares-Vásquez, Kamal Hossen, Hoang Dang, Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. Triaging incoming change requests: Bug or commit history, or code authorship? In *ICSM 2012*, pages 451–460. IEEE, 2012.

[12] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[13] Dominic Matter, Adrian Kuhn, and Oscar Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *MSR '09*, pages 131–140, May 2009.

[14] Ali Sajedi, Afsaneh Esteki, Ameneh GholiPour, Abram Hindle, and Eleni Stroulia. Involvement, contribution and influence in github and stack overflow. In *Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '14, Markham, Toronto, Canada, 2014. IBM/ACM.

[15] Ali Sajedi, Abram Hindle, and Eleni Stroulia. Crowdsourced bug triaging. In *ICSME '15*, Bremen, Germany, 2015. IEEE.

[16] Ali Sajedi, Vraj Shah, and Eleni Stroulia. Github's data adaptor, an eclipse plugin. In *Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '15, Markham, Toronto, Canada, 2015. IBM/ACM.

[17] Ramin Shokripour, John Anvik, Zarinah M. Kasirun, and Sima Zamani. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. MSR '13, pages 2–11. IEEE Press, 2013.

[18] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar Al-Kofahi, and Tien N. Nguyen. Fuzzy set-based automatic bug triaging (nier track). ICSE '11, pages 884–887. ACM, 2011.

[19] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In *ICSME 2014*, pages 181–190. IEEE, 2014.

[20] Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.