Fuzzy Set-based Automatic Bug Triaging (NIER Track)

Ahmed Tamrawi, Tung Thanh Nguyen, Jafar Al-Kofahi, Tien N. Nguyen
Electrical and Computer Engineering Department
Iowa State University
{atamrawi,tung,jafar,tien}@iastate.edu

ABSTRACT

Assigning a bug to the right developer is a key in reducing the cost, time, and efforts for developers in a bug fixing process. This assignment process is often referred to as *bug triaging*. In this paper, we propose Bugzie, a novel approach for automatic bug triaging based on fuzzy set-based modeling of bug-fixing expertise of developers. Bugzie considers a system to have multiple technical aspects, each is associated with technical terms. Then, it uses a fuzzy set to represent the developers who are capable/competent of fixing the bugs relevant to each term. The membership function of a developer in a fuzzy set is calculated via the terms extracted from the bug reports that (s)he has fixed, and the function is updated as new fixed reports are available. For a new bug report, its terms are extracted and corresponding fuzzy sets are union'ed. Potential fixers will be recommended based on their membership scores in the union'ed fuzzy set. Our preliminary results show that Bugzie achieves higher accuracy and efficiency than other state-of-the-art approaches.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Algorithms, Design, Reliability, Management

Keywords

Bug Triaging, Fuzzy Set

1. INTRODUCTION

Bug fixing is crucial in producing high-quality software products. When bug(s) are filed in a bug report, assigning it to the most capable and competent developer is important in reducing the cost and time in a bug fixing process [5]. This assignment process is referred to as *bug triaging* [1]. To help developers in this task, we propose Bugzie, a novel

ID:000002
FixingDate:2002-04-30 16:30:46 EDT
AssignedTo:James Moody
Summary:Opening repository resources doesn't honor type.
Description:Opening repository resource always open the default text editor and doesn't honor any mapping between resource types and editors. As a result it is not possible to view the contents of an image (*.gif file) in a sensible way....

Figure 1: Bug report 000002 in Eclipse project

ID:006021
FixingDate:2002-05-08 14:50:55 EDT
AssignedTo:James Moody
Summary:New Repository wizard follows implementation model, not user model.
Description:The new CVS Repository Connection wizard's layout is confusing. This is because it follows the implementation model of the order of fields in the full CVS location path, rather than the user model.

Figure 2: Bug report 006021 in Eclipse project

automatic bug triaging approach that models the bug-fixing tendency/expertise of developers with respect to technical aspects in a project based on their past fixing activity via fuzzy set theory [8], and then leverages such information to recommend most potential fixers for a new bug report.

Let us start with a motivating example on real-world bug reports. Figure 1 depicts a bug report from Eclipse project, with the relevant fields including a unique identification number of the report (ID), the fixing date (FixingDate), the fixing developer (AssignedTo), a short summary (Summary), and a full description (Description) of the bug.

The bug report describes an issue that the system always used its default editor to open any resource file (e.g. a GIF file) regardless of its file type. Analyzing the description, we found this report is related to a technical aspect in Eclipse, that is, *version control and management* (VCM) of software artifacts. The concept of VCM could be recognized in the report's content via its descriptive terms such as repository, resource, or editor. This technical function can be considered as project-specific since not all systems have it. Checking the corresponding fixed code in Eclipse, we found that the bug occurred in the code implementing an operation of VCM: opening a resource file in the repository. The bug was assigned to and fixed by a developer named *James Moody*.

Searching and analyzing several other Eclipse's bug reports, we found that James also fixed other VCM-related bugs, for example, the one in the report #6021 (Figure 2). That bug is related to VCM in which the function of repository connecting was not properly implemented. Thus, James Moody probably has the expertise, knowledge, or capability with respect to fixing the VCM-related bugs in the project.

Implications. The example suggests us the following implications for our approach:

1. A software system has several technical aspects. Each aspect is expressed via technical terms. A bug report is related to one or multiple technical aspects.

2. If a developer frequently fixes the bugs related to a technical aspect, we could consider him to have *bug-fixing expertise/capability* on that aspect, i.e., he could be a capable/competent fixer for a future bug related to that aspect.

We could determine the capable developers for the technical aspects in the system based on their past fixing activities. When a new bug is filed, we recommend the developers who are most capable of fixing bugs in the corresponding aspects.

2. APPROACH

There are two key research questions in Bugzie: *1) how to represent technical aspects of a system from software artifacts (e.g. bug reports)*, and 2) *given a bug report, how to determine who have the bug-fixing capability/expertise with respect to the reported technical aspect(s).*

We consider a technical aspect as a collection of technical terms that are extracted directly from the software artifacts in a project, and more specifically from bug reports. For the second research question, we utilize the *fuzzy set theory.* We use a *fuzzy set* C_t to represent the set of developers who have the bug-fixing expertise relevant to a specific technical term t, that is, the set of developers who are the most competent to fix the bugs relevant to the term t. The determination if a developer belongs to that fuzzy set is made via the occurrences of the terms in the bug reports that he has fixed. For a new bug report B with one or multiple technical aspects, the set C_B of capable developers toward B is modeled by a fuzzy set that is the union set of all fuzzy sets (over developers) corresponding to all terms associated with B.

Our algorithm has three main stages: 1) **training**: building fuzzy set C_t for each term t from available artifacts (e.g. fixed bug reports); 2) **recommending**: for a given unfixed bug report B, recommending a ranked list of developers capable of fixing it; and 3) **updating** the fuzzy sets when new information (e.g. new fixed bug reports) is available.

2.1 Training

In Bugzie, a fuzzy set C_t is determined via a membership function μ_t with values in the range of [0,1]. For a developer d, $\mu_t(d)$ determines how likely d belongs to the fuzzy set C_t, i.e. the degree to which d is capable of fixing the bug(s) relevant to t. We calculate $\mu_t(d)$ based on the correlation between the set D_d of bug reports d has fixed, and the set D_t of bug reports containing term t.

$$\mu_t(d) = \frac{|D_d \cap D_t|}{|D_d \cup D_t|} = \frac{n_{d,t}}{n_t + n_d - n_{d,t}}$$

In this formula, n_d, n_t, and $n_{d,t}$ are the number of bug reports that d has fixed, the number of reports containing the term t, and that with both, respectively (counted from the available training data, i.e. given fixed bug reports). The formula means that, if the more frequently a term t appears in the reports that developer d has fixed, the more likely that developer d has fixing expertise toward the technical aspects associated with t. The higher $\mu_t(d)$ is, the higher degree that d is a capable fixer for the bugs relevant to term t.

The value of $\mu_t(d) \in [0, 1]$. If $\mu_t(d) = 1$, then only d had fixed the bug reports containing t, thus, d is highly capable of fixing the bugs relevant to the aspects associated with term t. If $\mu_t(d) = 0$, d has never fixed any bug report containing t, thus, might not be the right fixer with respect to t.

The membership values within the interval [0,1] indicates the marginal elements of the class of developers defined by a term. Thus, the membership in a fuzzy set is an intrinsically gradual notion, instead of concrete as in conventional logic. That is, the boundary for the set of developers who are capable of fixing the bug(s) relevant to a term t is fuzzy.

2.2 Recommending

In this step, Bugzie recommends the most capable developers for each given unfixed bug report B. Since B reports on one or more technical aspects and those aspects could be recognized via the technical terms extracted from B, we consider the set of capable developers for B is a fuzzy union set C_B of all fuzzy sets corresponding to all the terms in B.

$$C_B = \bigcup_{t \in B} C_t$$

According to fuzzy set theory [8], the membership function of C_B is calculated as the following:

$$\mu_B(d) = 1 - \prod_{t \in B} (1 - \mu_t(d))$$

It could be seen that $\mu_B(d)$ is also within [0,1] and, by fuzzy set theory, it represents the degree in which developer d belongs to the set of capable fixers for the bug(s) reported in B. The value $\mu_B(d) = 0$ when all $\mu_t(d) = 0$, i.e. d has never fixed any report containing any term in B. Thus, Bugzie considers that d might not be as suitable as others in fixing technical issues reported in B. Otherwise, if there is a term with $\mu_t(d) = 1$, then $\mu_B(d) = 1$ and d is considered as the capable developer (since only d has fixed bug reports with term t before). In general cases, the more terms in B have high $\mu_t(d)$ scores, the higher $\mu_B(d)$ is, i.e. the more likely d is a capable fixer for bug report B.

After calculating $\mu_B(d)$ for all available developers, Bugzie ranks them based on those membership values and recommends the top-n developers as the ones who should fix the bug(s) reported in B.

2.3 Updating

When new information is available (e.g. new bug reports are fixed by some developers), Bugzie updates its training data by updating all existing fuzzy sets C_t and creating new sets for new terms. The update process can be done *incrementally.* As we could see, C_t is defined via the membership values $\mu_t(d)$s, and $\mu_t(d)$ is calculated via n_d, n_t, and $n_{d,t}$. Therefore, Bugzie stores only the values n_d, n_t, and $n_{d,t}$, and updates them when there are newly available fixed bug reports by adding new corresponding counts for the new data. Specifically, if a new term (or a new developer) appears in new data, Bugzie just creates new counting numbers n_t (or n_d) and $n_{d,t}$s. If a developer has a new fixing activity, Bugzie just updates the corresponding counting numbers n_d and $n_{d,t}$s. For example, the number of reports fixed by d is updated with the number of new fixed reports from d: $n_d := n_d + n'_d$. Other derivative values such as $\mu_t(d)$ and $\mu_B(d)$ are calculated from those counts on demand. This makes our incremental training algorithm very efficient in comparison with other modeling/learning techniques. Importantly, it fits well with the evolutionary nature of a software system and a software development process.

3. EVALUATION

3.1 Experiment Setup

We conducted a preliminary evaluation on Eclipse project, which has been used in evaluating the existing state-of-the-art approaches [1, 3, 7]. From Eclipse's bug tracking repository [6], we collected 69,829 bug reports that have been filed and fixed from January 2008 to November 2010. For each bug report, we extracted its unique ID, the actual fixing developer's ID, short summary, and full description. There are in total 1,510 fixing developers for those bug reports.

We merged the summary and description of each bug report, extracted their terms and preprocessed them, such as stemming for term normalization and removing grammatical and stop words. Finally, we had a total of 103,690 terms.

We used the same longitudinal experiment setup as in [3], simulating the usage of our tool in reality. That is, all bug reports are sorted in the chronological order, and then divided into 11 non-overlapped and equally sized frames. Each frame is indexed corresponding to their creation time.

Initially, frame 0 with its bug reports are used for training only. Then, Bugzie uses that training data to recommend for the first 100 bug reports in frame 1. Bugzie gives a top list of T developers recommended to fix each of those 100 bug reports. If the recommendation list for a bug report B contains its actual fixer, we count this as a hit (i.e. a correct recommendation). After that, we update the counts for the fuzzy sets with the tested 100 bug reports and move to the next 100 bug reports in the same frame.

After completing frame 1, the updated training data is then used to test frame 2 in the same manner. For each frame under test, we use the prior frames for training, and calculate the *prediction accuracy* as in [3], i.e. the ratio between the number of hits over the total prediction cases. We then calculate the average value on all 10 frames. An average value is calculated for each selection of the top-ranked list of T from 1-5. We repeat for the remaining frames.

For the comparison purpose, we also used Weka [12] to re-implement the existing state-of-the-art approaches [1, 7, 3] with the same experimental setup and with the descriptions of their approaches in their papers. We calculated the prediction accuracy and measured time efficiency, which is the total time of training, updating, and recommending.

3.2 Results

Table 1 shows the results from different approaches. Anvik *et al.* [1] employed SVM, Naive Bayes, and C4.5's classifiers. Bhattacharya and Neamtiu [3] used Naive Bayes and Bayesian network with and without incremental learning. Cubranic and Murphy [5] used Naive Bayes.

Figure 3 displays the accuracy comparison of Bugzie with others when the recommended list contains 1 to 5 top-ranked developers. As seen, Bugzie outperforms other approaches both in term of prediction accuracy and time efficiency. For top-1 recommendation (i.e. recommending only one fixer for each bug report under test), Bugzie has a prediction accuracy of 37.81% on average, i.e. on average in 37.81% of the cases, it correctly recommends the developer who actually fixed the bug(s). For top-5 recommendation, it has accuracy 68%, i.e. in 68% of the cases, the actual fixer was in its top-5 recommended list. Other machine-learning approaches reach the maximum accuracy of 53.02% at their top-5 recommendations.

Importantly, Bugzie is also more time efficient than those approaches. Table 2 shows total time in hours spent in training and recommendation for different approaches. As shown, the training time of Bugzie (4 hours) is smaller than that of other approaches. The corresponding time of the second fastest approach is about 6 hours. For the case of C4.5, our data set is too large for Weka to run on all 11 frames. The value in Table 2 at C4.5 line is only for 4 frames. The recommendation time is much smaller because Bugzie just needs to compute $\mu_B(d)$ (Section 2.2) and ranks developers based on their scores. That is, Bugzie is more efficient in computation, while other approaches that use machine-learning techniques may not scale well for very large data sets.

Table 1: Prediction Accuracy Result (%)

Approach	Top-*1*	Top-*2*	Top-*3*	Top-*4*	Top-*5*
Naïve Bayes	23.68	33.72	39.76	43.88	47.05
Bayesian Network	12.20	18.03	22.17	25.50	27.88
C4.5 (Decision Trees)	18.68	23.97	24.86	25.10	25.14
SVM	27.38	38.53	45.26	49.78	53.02
Inc Naïve Bayes	25.86	36.39	42.49	46.61	49.78
Inc Bayesian Network	14.06	20.91	25.52	29.01	31.86
Fuzzy Set (this paper)	37.81	52.11	59.70	64.52	68.00

Table 2: Time Efficiency Comparison (hh:mm:ss)

Approach	Training	Recommendation
Naïve Bayes	08:49:17	131:33:04
Bayesian Network	15:21:38	180:26:58
C4.5 (Decision Trees)	129:17:37	00:05:58
SVM	06:01:57	11:46:09
Incremental Naïve Bayes	36:52:43	129:15:49
Incremental Bayesian Network	53:47:42	190:16:47
Fuzzy Set (this paper)	03:52:57	00:00:22

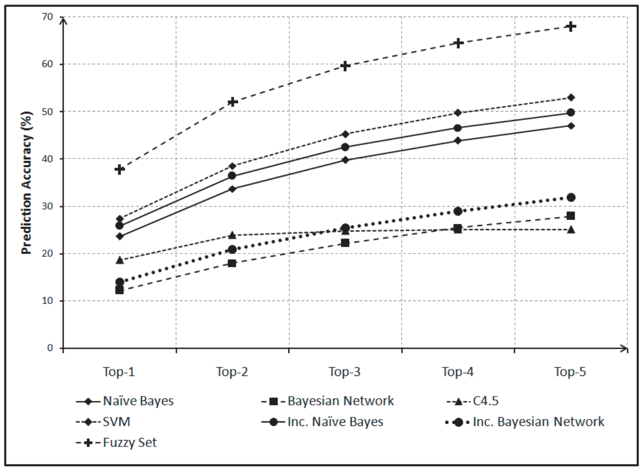

Figure 3: Prediction Accuracy Comparison

4. RELATED WORK

There are several approaches that apply machine learning (ML) and/or information retrieval (IR) to (semi-)automate the process of bug triaging. The first approach along that line is from Cubranic and Murphy [5]. From the titles and descriptions of bug reports, *keywords* and *developers' IDs* are extracted and used to build a text classifier using *Naive Bayes* technique. Their classifier will recommend potential fixers based on the classification of a new report. Their prediction accuracy is up to 30% on an Eclipse's bug report data set from Jan to Sep-2002. Anvik *et al.* [1] also follow similar ML approach and improve Cubranic *et al.*'s work by filtering out invalid data such as unfixed bug reports, no-longer-working or inactive developers. With 3 different classifiers using SVM, Naive Bayes, and C4.5, they achieved a precision of up to 64%. In contrast to the ML techniques in those approaches, our fuzzy set approach has higher computational efficiency in *incremental data training*. Moreover, it is able to naturally provide a ranked list of potential fixers, while the outcome of a classifier in their approaches has the assignment of a bug report to one specific developer.

Another related approach is from Bhattacharya and Neamtiu [3]. Similar to Bugzie, their model is capable of incremental learning. However, in contrast to fuzzy set approach in Bugzie, they use a ML approach with Naive Bayes and Bayesian network to build classifiers for keywords extracted in reports. Therefore, Bugzie has better time efficiency and a more natural ranking scheme than their ML classifiers. Moreover, as seen in Section 3, Bugzie outperformed their (incremental) Naive Bayes and Bayesian classifiers.

The idea of *bug tossing graphs* was first introduced by Jeong *et al.* [7] in which their Markov-based model learns the patterns of bug tossing from developers to developers after a bug was assigned in the past, and it uses such knowledge to improve bug triaging. Their goal is more toward reducing the lengths of bug tossing paths [7], rather than addressing the question of who should fix a particular bug as in an initial assignment. We can combine fuzzy set approach with the use of bug tossing graphs to further improve our accuracy.

Lin *et al.* [9] use a ML approach with SVM and C4.5 classifiers on both textual data and non-text fields (e.g. bug type, priority, submitter, phase and module IDs). Executing on a proprietary project with 2,576 bug records, their models achieve the accuracy of up to 77.64%. The accuracy is 63% if module IDs were not considered. Bugzie has higher accuracy and could integrate non-text fields for further improvement.

Other researchers use IR for automatic bug triaging. Canfora and Cerulo [4] use the terms of fixed change requests to index source files and developers, and then query them as a new change request comes in order to automate bug triaging. However, the accuracy was not very good (10-20% on Mozilla and 30-50% on KDE). Their indexing scheme does not support incremental learning and probability.

Matter *et al.* [10] introduce Develect, a model for developers' expertise by extracting terms in their contributed code. Then, a developer's expertise is represented by a vector of frequencies of terms appearing in his source files. The vector for a new bug report is compared with the vectors for developers for bug triaging. Testing on 130,769 bug reports in Eclipse, the accuracy is not as high as Bugzie (up to 71% with top-10 recommendation list, respectively). While Develect is based on vector-based model (VSM), a deterministic traditional IR method, Bugzie models developers' with fuzzy sets, enabling more *flexible computation and modeling* of developers' bug-fixing expertise, as well as enabling *incremental learning* for time efficiency. For example, in Develect, the length of a vector representing a developer's expertise must cover all terms occurring in the data set. With the fuzzy set nature, in Bugzie, thresholds are chosen to be more selective in a set of terms for one developer. Moreover, with evolving software (new developers and terms), VSM must recompute entire vector set. Baysal *et al.* [2] proposed to enhance VSM in modeling developers' expertise with preference elicitation and task allocation. Rahman *et al.* [11] measure the quality of assignment by the match between requested (from bug reports) and available (from developers) competence profile.

In brief, existing ML-based classification approaches [1, 3, 9] characterize the classes of bugs that each developer is capable of, and then classify a new bug report based on that classification for bug triaging. Other approaches aim to profile developers' expertise via terms in past fixing bug reports, and match a new report with such profiles [2, 10].

5. CONCLUSIONS

In this paper, we propose Bugzie, a new fuzzy set-based approach for automatic bug triaging. Fuzzy sets are used to represent the sets of capable developers of fixing the bugs related to individual technical aspects via technical terms. Such fuzzy sets are computed for each term in a new bug report and then are union'ed to find capable developers for the report. Preliminary evaluation shows that Bugzie achieves higher accuracy and efficiency than existing approaches.

Acknowledgment. This project is funded by NSF CCF-1018600 award.

6. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In ICSE '06, pages 361–370. ACM, 2006.

[2] O. Baysal, M. W. Godfrey, and R. Cohen. A bug you like: A framework for automated assignment of bugs. In ICPC'09, pages 297-298. IEEE CS, 2009.

[3] P. Bhattacharya and I. Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *ICSM'10*. IEEE CS, 2010.

[4] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In SAC'06: ACM symposium on Applied computing, ACM Press.

[5] D. Cubranic and G. Murphy. Automatic bug triage using text categorization. In SEKE'04. KSI Press.

[6] Eclipse bugzilla repository. bugs.eclipse.org/bugs/.

[7] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *FSE'09*, ACM.

[8] G.J. Klir and Bo Yuan. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall, 1995.

[9] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang. An empirical study on bug assignment automation using Chinese bug data. In *ESEM'09*. IEEE CS, 2009.

[10] D. Matter, A. Kuhn, and O. Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *MSR'09*, pp. 131–140. IEEE CS, 2009.

[11] M. Rahman, G. Ruhe, T. Zimmermann. Optimized assignment of developers for fixing bugs: an initial evaluation for Eclipse projects. In ESEM'09, IEEE CS.

[12] Weka. http://www.cs.waikato.ac.nz/ml/weka/.