APTIKOM Journal on Computer Science and Information Technologies
Vol. 1, No. 1, 2016, pp. 1~11
ISSN: 2528-2417
DOI: 10.11591/APTIKOM.J.CSIT.6

Survey Based Classification of Bug Triage Approaches

Asmita Yadav*, Sandeep Kumar Singh
Department of Computer Science, Jaypee Institute of Information Technology (JIIT), Noida, UP, India
*Corresponding author, e-mail: asmita.yadav85@gmail.com

Abstract

This paper presents a comprehensive survey of bug triaging approaches in three classes namely machine learning based, meta-data based and profile based. All approaches under three categories are critically compared and some potential future directions and challenges are reported. Findings from the survey show that there is a lot of scope to work in cold-start problem, developer- profiling, load balancing, and reopened bug analysis.

Keywords: *Mining Software Repository, Bug Report, Bug report Triger, bug summarization, Machine-learning approach, developer profile, Bug report Metadata*

1. Introduction

All the valuable information about software project is stored in the software repositories like: Bugzilla for bug tracking system, Jira as task repositories etc. Large number of bugs is introduced during the software development are maintenance due to poor design, wrong implementation strategy, misunderstood requirements. In a bug repository, bugs can be reported by any user, developer, or a technical support team member [1]. Each new bug must be assigned to a relevant developer for fixing. For this process, a Triager needs to analyze bugs carefully for validity, duplicity, importance and potential fixer.

To find a relevant developer who has required experience in fixing similar bugs is a difficult task for a triager [2]. A number Triaging approaches have been proposed in past few years. These can be broadly categorized into three categories: machine learning based, profile based, and Metadata based as shown in Table 1. The approaches discussed under these categories still have scope of improvements [3]. This paper discusses a comprehensive survey on all different bug fixing approaches, their major drawbacks or limitations, and also give some new direction for future research.

Rest of the paper is organized as follows: section 2 presents the background of bug reports terminology, life-cycle of the bug report, bug report summary and ranking process. Section 3 presents a survey based on recommendation system, and a comparison on recommendation is described in section 4. Sections 5 discuss some research challenges related to recommendation system and section 6 concludes the paper.

2. Background

This section explains some basic concepts related to the bug report and bug-life cycle.

2.1. Bug- report Terminology

2.1.1. Bug Repository

All the software projects use different forms of repositories like: software control repositories, code repositories, bug repositories, and archived communication etc[4,5]. Most of the bug repositories (Example: Jira, Bugzilla) are of open source software projects to store the bug report information that can be filed by any of the end-users or developers.

2.1.2. Bug

If an unexpected result is produced by the computer program or a system, then it called software bug, fault or error. Sometime, an issue can be a bug, but it's not always compulsory. There is a difference

Received Dec 09, 2015; Revised Feb 03, 2016; Accepted Feb 21, 2016

between bug and an issue. Issues can be a request of a feature or a task raised by a developer or end-user whereas a bug can be an error that produced some unexpected or incorrect result.

2.1.3. Bug – report

Details of a software bug are stored in bug reports. Each bug report has unique identification number. A bug report contains lots of information related to bug like: bug title, summary of bug, bug reported time, time when bug modified, bug version, bug product and component name, name of the assignee, who fix up the bug, bug resolution status (i.e. bug is new, unconfirmed or resolved), developers comments and bug severity[6].

2.1.4. Bug- repot Triage

To resolve a new bug, each bug must be assigned to a relevant developer who has an appropriate experience in restoring similar types of bug. Bug assignment process can be done manually, which becomes labor intensive, error-prone and time- consuming. To reduce the time and cost of bug assignment process, the first automatic bug triager was proposed by Cubranic and Murphy[7]. Thereafter, many automatic bug triage approaches were proposed that are based on machine learning [8-16], meta-data [17-23], or developer profile [24-27]. These are shown in Table 1.

Table 1. References for all Approaches

Triaging Category	Paper Title
Machine Learning Based Approach	Assigning change requests to software developers[8]
	Automatic assignment of work item[9]
	Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions[10]
	Highly-accurate Bug Triage using Machine Learning[11]
	Improving bug triage with Bug tossing Graphs[12]
	Automated, highly-accurate, bug assignment using machine learning and tossing graphs[13]
	An Approach to Improving Bug Assignment with bug tossing graph and bug similarities[14]
	Novel metrics for bug triage[15]
	Automatic Bug Triage using Semi-Supervised Text Classification[16]
Meta-Data Based Approch	COSTRIAGE: A Cost-Aware Triage Algorithm for Bug Reporting[17]
	A time based approach to Automatic Bug Report Assignment[18]
	Topic- based, time aware bug assignment[19]
	Improving automatic bug assignment using time- meta in term weights[20]
	Effective Bug Triage based on Historical Bug-Fix information[21]
	Automatic Bug Assignment Using Information Extraction Methods[22]
	A Noun based approach to feature location using time aware term- weighting[23]
Profile Based Approach	An Automated Bug Triage Approach: A Concept Profile and Social network Based Developer Recommendation[24]
	Bug report assignee Recommendation using Activity Profile[25]
	A Hybrid Bug Triage Algorithm for Develop recommendation[26]
	Efficient Bug Triaging Using Text Mining[27]

2.1.5. Bug- report Duplication

A newly reported bug in issues tracking system can be a duplicate bug that has the same root of a master or existing bug. Duplicate bug can either be originated from the same root source as existing bug but may have a different failure or have same description of the same failure as an existing bug [28]. However in practice, duplicate bugs can be avoided only when the developer knows about all the existing bugs, which is practically not possible. An important task of a bug triage is to detect duplicate bugs and remove it in order to save the time for developers to fix the bug and reduce triaging cost.

2.1.6. Bug Tracking System

All bug reports that are either reported by the developers, end user or fixed by the developers are stored in a bug tracking system (BTS). BTS is also called issue tracking system. These bugs tracking systems (like: Bugzilla, Jira etc) are used by various open source projects (like: Mozilla, Eclipse etc) help to manage the bugs.

2.1.7. Bug Report Prioritization

A difficult and time consuming task for bug repositories is to host the large number of newly submitted bug reports. To resolve this problem, developer may assign bug priority (P1, P2, P3, P4 and P5) based importance of bugs in a system. Various bug priority recommendation are proposed using SVM and Naïve Bayes classification [1,10, 35].

2.2. The Life-cycle of a Bug Report

A bug report has various resolution statuses over its lifetime that is depicted in Figure 1. Every newly submitted bug in repository has a category Unconfirmed. After Verification, if bug is not duplicate then it status is given as New. After checking that bug is genuine, it is assigned to a relevant developer for fixing and its status converted as 'Assigned'. Assigned Developer try to resolve bug and when the bug has been resolved by the assignee, then it is given the status as resolved. Bug can be Reopened in the case, when the end user or tester is not satisfied with the provided solution of the bug and again assigned to Assignee to fix it. If the tester is satisfied with the solution then its status is changed to 'Verified'. Final status of bug is closed.

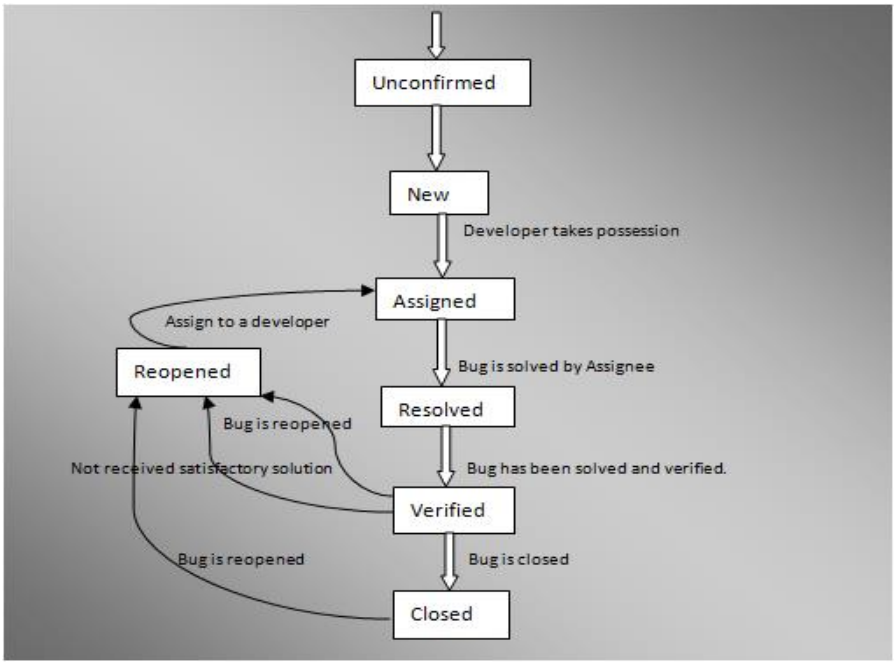

Figure 1. Bug –report Life Cycle

2.3. Bug Report Summarization

All the substantial details of reported bugs are stored in the bug repository that can help to understand the entire bug history. Bug summarization techniques helps to summarize entire bug reports, in order to decrease time and cost of triager and increase the accuracy of Triager. In paper [29][30], author has reported some approaches to reduce the bug data by collecting only meaningful information from the bug reports using either feature & instance reduction techniques or bug summarization techniques.

3. Bug Triaging Approaches

An automated process which can predict a relevant developer to fix new bug by using previous and resolved bugs' history, bugs metadata or developer-profile, is called Bug Triaging process Figure 2 presents an overview of automated bug triaging system.

Previously resolved bug reports data (that can contain bug metadata, bug description or bug comments) and developer details are captured by recommendation engine as an input. A similarity matrix

is prepared by using any of the similarity matrix techniques (i.e. td-idf, jaccard index, cosine, overlap coefficient etc). It then produces a developer's ranklist, and finally predicts a relevant developer who can fix the bug. A feedback is send to both bug report history as well as to predicted developer.

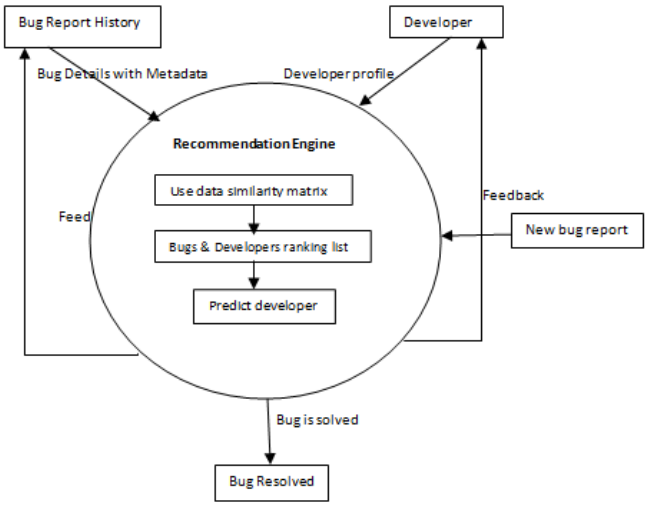

Figure 2. Automated Bug Triaging System

3.1. Bug Triaging Techniques

3.1.1. Machine Learning Based Approaches

Various bug triaging approaches are based on machine learning techniques for assigning a bug report to an experienced developer who has enough knowledge to fix the bug. In these techniques, previously resolved bug reposts are used as an input to train a classifier, and then this trained classifier classify and assign new bug report to relevant developer. According to survey, first Machine learning approach is presented by Cubranic and Murphy [31] which is based on bug reports. Machine learning techniques can be categorized into three types namely supervised learning, unsupervised learning and reinforcement learning. In our survey, we only focused on supervised learning approaches that contain various potential algorithms like: Naïve-bayes, support vector machine, tossing graph, vector space model etc.

In 2009, the bug tossing concepts was defined and described by Jeong [12]. According to author's survey, in Mozilla and Eclipse, around 37%-44% of bug reports are tossed again that can be reduced up to 72% and improve 23% automatic prediction accuracy by using Jeong tossing model. This is a first work on tossing graph that explained use of a basic classifier without considering the inter-feedback process and developer activity. Furthermore, Pamela[13] extended their work to remove some of these limitation by using a fine-grained, multi –features tossing graph (Product, Component and activity days are extra attributes to an edge) with intra update, which is able to improve accuracy upto 86.09% and reduce the tossing path lengths by up to 83.28% in Eclipse and 86.67% in Mozilla. This approach is not applicable on small projects due to their limited numbers of bug features on the node of multi tossing graphs i.e. product, component and activity. This approach is not be able to handle the developer load balancing problem. To remove Jeong[12] limitations, Liguo [14] also proposed an approach by using both graph tossing and vector space model. To measure the path between the assignee and developer, they used weight based breadth first algorithm. Up to 84% bug tossing length can be reduced by using this approach. Although, they only evaluate their approach on two open source projects (Eclipse & Mozilla) by considering very few bug report features for similarity measure. Success of this approach in closed source project needs to be improved. V.Akila[15] has proposed an approach based on bug tossing graph by using metric to reduced hop path and route a bug to the correct developer in the best (optimal) route. In this respect, Levenshtein similarity achieved best correlation coefficient value (.9714 for 30% data set and .9671 for 20% data set) with respect to precision. But this approach has no indicators

for measuring the strength of the retrieved paths, if the extracted paths of more than one developer have same distance to the original path.

Jifeng [16] presented another mechanism of machine learning techniques to combine the naïve bayes classifier and expectation- maximization (EM) that utilized both labeled and unlabeled bug reports. They reported improved classification accuracy up to 6 % only by this classifier. It could not achieve sufficient improvements in naïve bayes classifier. So that, obtain experimental results are not up to mark for the real-world applications. This may be due to use of inappropriate bug reports, wrong assumption of EM model for real- world data, and selection of irrelevant developer as a bug fixer.

Jonas[9] proposed a semi- automatic assignment model approach that is based on unified- model and presents a relevant artifact by using the history of bug reports and association between work item (newly reported bug) and developer. At a certain time, a snap-shot of the project is captured to assign all the work items that finally have a fixed state. Two classifiers: SVM (support vector machine) and Naïve bayes both gives better results for certain project state(state-based: snap shot of a project at a certain time). For datasets UNICASE, DOLLI and King Tale, SVM classifier produced 38%, 28.9%, 37.4% accuracy and for the same database, naïve bayes achieved 39.1%, 29.7%, 37.8% accuracy respectively. Although, that model based approach is only applicable on those work items that are linked to the functional requirement of the work item and is therefore not directly applicable in scenarios where links do not exist.

Although, various techniques are proposed for concept- location which is based on the search of abstract system dependence graphs, like: static as well as dynamic based techniques, IR- based techniques etc. but no work has applied concept location techniques to the problem of expert developer recommendation. In 2011, concept location approach has been proposed by Huzefa [8]. Here, author presents an approach to recommend a ranked list of expert developers with relevant source code. This source-code corpus is created from extracted comments and identifiers from the source-code and is indexed by Latent Semantic Indexing (LSI). xFinder and xFactor is used for expert developer recommendation to prepare the rank-list of the relevant developers. Three open source system, namely KOffice, Eclipse and ArgoUML are used for evaluation and accuracy of 95%, 82% and 80% respectively is recorded by this recommendation system. xFinder is used to increase the rank effectiveness and in 50% of cases, the first relevant developer is found in the first position of the rank-list. This rank-list is prepared at the four granularity level i.e. file, package, system and overall. In few cases, location based tools does not return exactly relevant source-code or class file for bug fixing. 88.89% request-level of accuracy is achieved. Developer's expertise, knowledge and experience are only extracted from his/ her previous contribution for bug fixing; still this is not a sufficient condition to correctly assign developer to resolve the bug because in some cases, a developer resolved more than bugs , then he have numbers of committer' Ids. There is a challenge to select only one committer' Id for developer identification by xFinder.

3.1.2. Meta- Data Based

Bug metadata is used by many researchers to propose methods for automatic bug assignments. Metadata has all types of bug related details like: bug time stamp (means when bug is filed, when it tossed between the developers for fixing and final bug fixing time), bug history, bug comments, developers sparseness etc.

Term weighting technique is mostly used by the researchers to determine the term frequency. This textual information is used to prepare rank-list of relevant developers for bug fixing. Although, the term frequency- inverse document frequency (tf-idf) is common technique for bug assignment that is explained by calvalcanti [32]. But, Tf-Idf does not consider time- stamp, i.e. the time of using a term. Ramin & Anvik [18] presented an approach ABA-Time-tf-idf (Automatic Bug Assignment using the Time–tf-idf) term weighting technique. They considered time when the terms were used by developers to assign weights to terms during triaging process. All the important information like: time difference between the last activity of the developer and the new bug' reporting date, time spent for fixing previous bug is extracted from the time- metadata to decide the developer ranking. The results of ABA-Time-tf-idf approach had indicated an improvement between 26-37.2%, 3.4-14.4%, 5.6-17.2% and 12.6-19.8% in comparison to the average accuracies of SVM, NB, VSM, SUM approaches respectively on five random data-sets of Eclipse projects. This methodology assumes that the developer who committed the changes to repository is the actual fixer of the bug report. In some of the projects, few of the developers' works as a gate keepers who have only the permission to commit to the source code, that means developer who changes the source code and who committed the change(s) in the software repository, can be different.

Another improved version in time stamp is presented by Ramin & Anvik [20] developer' recommendation system based on the similarity (weight) between the new bug' term and previous bug-report information (corpus). They considered important details from a bug like: the term creation, modification time and who changed it. This information is used for calculation of the developer activity at various periods of project's life. If a developer A used a term 'switch' for bug fixing, two years ago and developer B, used it 8 month ago on another bug, containing the same term. Then developer B is more appropriate to fix the new bug that contains the same term 'switch'. The TNBA (time –aware noun- based bug assignment) approach outperformed the TNBA (no-time), tf-idf and VSM (time) approaches by as much as 14, 12 and 48% on Eclipse, Netbeans and ArgoUML projects respectively. This approach only worked on the exact matching in text processing and cannot handled approximation matching by using ontology to handles the synonyms of the bug text.

All the information like: developer identifier, time-stamp and commit comments are stored in the metadata. Only Sisman & kak [33] used the time-metadata for feature location approach. Sima, Sai & Ramin [23] has proposed an approach that included weighting and ranking the source- code locations based on both the textual similarity with a change request and the use of the time-metadata. It used only the noun terms for weighting to reduce the dataset volume. This approach gives much better results as compared to the tf-idf techniques by up to 15%, 10% and 14 % in terms of accuracy, effectiveness and performance respectively.

Another approach for automatic bug assignment is given by Ramin [22] by using information extraction from bug metadata. In this, they calculated the similarities between bug reports and commits and then found the similar phrases that are used to link a bug report to a specific commit and finally determine the exact location of new bug with relevant developer to fix a new bug. For experiential work three open sources are considered namely Eclipse, Mozilla and Gnome, and they received 62%, 43% and 41% recall levels respectively.

Mamdouh [27] proposed an bug triaging approach based on text mining concepts by using five term selection methods (Log odds ratio, Chi-square, Term frequency relevance frequency, Mutual information and Distinguishing feature selector) to predict an experienced developer to resolve bug. According to this approach, X2 gives better results as compare to other selection methods in terms of F-score. It improved the F-score by 6.2%, 38.2% 26.5% and 12.1% for all open source system Eclipse-SWT, Eclipse-UI, Netbeans, and Maemo respectively.

The historical data extractions information for bugFixer has been used by Hao [21] to construct a developer – component- Bug network (DCB). For this DCB network, they established a relationship between the developer and source code component and also found the relation between source code component and bugs. It then calculated the similarity among new bug and existing bugs. This approach correctly ranked the bugs in Eclipse by up to 42.36% for first recommendation list. This approach suffered cold start problem when new bug or developer have no previous and historical information.

In 2014, Tung [19] worked on a problem to determine amount of time required to fix a bug. Although, this problem is also mentioned and discussed by Jin [17] in reference of bug triaging issue. Here, they focused only to achieve better accuracy and cost without considering the time- complexity of the problem. They evaluated their approach on four different open source project namely, Apache, Eclipse, Linux kernel and Mozilla and achieved better accuracy by reducing 30% cost of the triage. However, they have no solution to handle bug resolution time. Then Tung [19] proposed a model that is a topic- based, log- normal regression model (combination of CosTriage and Regression model) that can predicate the resolution time of a given bug , if it is already assigned to a given developer.

3.1.3. Profile Based

Who can fix the bug? Is an important question for bug recommendation system. To find a relevant developer for bug fixing is a major research issue and various approaches are proposed for this problem only. Here, we only survey few of such recommendation approaches that are based on automated developer profile. In 2012, Tao Zhang [24] has proposed a developer profile concept by using related bug reports concepts to prepare a developer social network. An expert developer rank list with bug fixing cost is maintained, and in addition it also record active and inactive developers.

A hybrid bug triage approach is presented by Tao [26] for developer recommendation. In this developer ranking is calculated by embedded system i.e. experience and probability model. Here experience model has all the details of all fixed bugs by the developer with bug fixing cost, and it also added a new feature re-opened bugs to get the developer' probability of fixing bug. This approach provides a high F-score for JBoss and Eclipse upto 71%, and 67% respectively, but this approach is not

effective on business projects. This approach also suffers from cold start problem, i.e. in case of a new type of bug or a new developer none of the historical data is helpful to identify the relevant features.

To improve the bug triage accuracy, an activity profile has been described by Hoda[25]. In that method, developer ranked list is based on the developer expertise in the bug report topic. It has a new way to save bug fix time by recommending and adding all new developer who are willing to volunteer and have sufficient knowledge for bug resolving. After experiment, this proposed approach has better hit ratio i.e. 88% as compare to LDA and SVM-based activity profile techniques. Although, to improve the accuracy of the Triager, an ensemble classifier (like: SVM+LDA) can be a better option as compared to use a single classifier.

4. Comparative Assessment of Survey Paper

A comparison between all approaches is shown in Table 2 in terms of features used in triaging approaches. These features are: DC-Data collection, PreP-Preprocesing, SM-Similarity measure, SC-Single classifier, MC-Multi classifiers, DP-Developer Profile, RC-Reduce cycle, RS-Reduce Sparseness , LB-Load balancing, DAvgT-Developer Average time, BP- Bug priority, DS- Developer specialization, DA-Developer Activity, MD-Metadata. Few of the papers are worked on developer sparseness, developer activity and developer specialization, although these are most important parameters for building the developer profile. Meta data is also not referred by the authors. These all issues give challenges for the researchers in terms of cold start problem, activity profile, load-balancing etc.

Bug triaging approaches presented in previous section have some advantages as well as disadvantages. In Machine learning (ML) approaches, there is a possibility that developer may not have complete domain knowledge for bug solving. To achieve better accuracy, a feedback system can be used, so that all resolved bugs history is updated, which can be used by the classifier for better recommendation. ML approaches have a limitation that it does not consider a new developer as a potential bug fixer.

Another bug fixing technique that considered all the details (including previous bug history) of bug reports, named metadata based approach, which we use bug information like: bug basic data, all bug related comments and logs that filed by another user, bug long description etc. But still this approach has a problem to select a relevant developer, who can fix the new bug that has purely different content from the previously fixed bugs.

Although, Profile based approach is better in the case of developer selection as compared to ML and metadata approaches as it can reduce developer sparseness. If a new developer or new type of bug report is reported and classifier have not found any previous and historical information for verifying the relevancy, then this approach does not consider a new developer as a potential bug fixer for newly reported bug.

Table 2. Features comparsion between all Traiging approaches

Paper Name	Features													
	DC	Pre.P	SM	SC	MC	DP	RC	RS	LB	DAvgT	BP	DS	DA	MD
Assigning change requests to software developers	Y	Y	Y	Y	N	Y	Y	N	N	N	N	Y	N	N
Automatic assignment of work item	Y	Y	Y	N	Y	N	N	N	N	N	N	N	N	N
Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions	Y	Y	Y	N	Y	N	Y	N	N	N	Y	Y	N	N
Highly-accurate Bug Triage using Machine Learning	Y	Y	Y	Y	N	N	Y	N	Y	N	N	Y	N	N
Improving bug triage with Bug tossing Graphs	Y	Y	Y	Y	N	N	Y	N	N	N	N	Y	N	N
Automated, highly-accurate, bug assignment using machine learning	Y	Y	Y	Y	N	N	Y	N	Y	N	N	Y	Y	N

and tossing graphs														
An Approach to Improving Bug Assignment with bug tossing graph and bug similarities	Y	Y	Y	Y	N	N	Y	N	N	N	N	Y	N	N
Novel metrics for bug triage	Y	Y	Y	Y	N	N	Y	N	N	N	N	N	N	N
Automatic Bug Triage using Semi-Supervised Text Classification	Y	Y	Y	Y	N	N	N	Y	N	N	N	Y	N	N
COSTRIAGE: A Cost-Aware Triage Algorithm for Bug Reporting	Y	Y	Y	Y	N	N	N	Y	N	N	N	Y	N	Y
A time based approach to Automatic Bug Report Assignment	Y	Y	Y	N	Y	N	Y	Y	N	N	N	Y	N	Y
Topic- based, time aware bug assignment	Y	Y	Y	Y	N	N	N	Y	Y	N	N	N	N	Y
Improving automatic bug assignment using time- meta in term weights	Y	Y	Y	N	Y	N	Y	N	N	N	N	Y	N	Y
Effective Bug Triage based on Historical Bug-Fix information	Y	Y	Y	Y	N	N	Y	N	N	N	N	Y	N	Y
Automatic Bug Assignment Using Information Extraction Methods	Y	Y	Y	Y	N	N	N	N	N	N	N	Y	N	Y
A Noun based approach to feature location using time aware term-weighting	Y	Y	Y	N	Y	N	Y	N	N	N	N	Y	N	Y
An Automated Bug Triage Approach: A Concept Profile and Social network Based Developer Recommendation	Y	Y	Y	Y	N	Y	Y	Y	N	N	N	Y	N	N
Bug report assignee Recommendation using Activity Profile	Y	Y	Y	Y	N	Y	Y	N	N	N	N	Y	Y	N
A Hybrid Bug Triage Algorithm for Develop recommendation	Y	Y	Y	Y	N	Y	Y	N	N	N	N	Y	N	N
Efficient Bug Triaging Using Text Mining	Y	Y	Y	Y	N	Y	Y	N	N	N	N	Y	N	N

Note: - Y-Yes, N- No

5. Future Direction for Research

This section presents several reason and challenges in bug triaging approaches. Which are:-

5.1. Cold-Start Problem

When a new developer, new bug report or a new system enters into the recommendation system, then a problem is accrued called cold start problem. Profile based technique can not recommend a new developers, who wants to fix a bug report, without having any stored preference and previous rating

history [34]. This type of situation is also faced in the case of new bug report, which has no historical information and data for solving and verifying the newly reported bug.

5.2. Developer-Activity

The major problem that encountered in the recommendation system is developer sparseness, means developers activities is scattered at a widely spaced intervals, examle: If a developer is inactive for more than 100 days or he/she has resolved less than 10 bugs. Then, those developers are considered as an inactive or retired developers and none of the bug is assigned to them although such developers may be interested to fix the bug. So there is a need to design a better heuristics to differentiate active from inactive developers to reduce the sparseness of previously fixed bugs.

5.3. Developer Weight Analysis

At the time of developer' ranking for triaging, exiting approaches have only considered those developers who have finally fixed the bug. However, it is common that some developers also contribute partially to the final patch in various ways (mailing list, comments etc). Such developers' contributions need to be incorporated in ranking process. To manage this problem, a developer social network can be created that list all the relevant developers who can fix the particular bug report.

In some project, few developers work as 'gate keeper'', who only have the necessary permissions to commits to the software repository and these commits can be used by other software projects. In some of the cases, people who commit the changes into the source repository are different from the people who actually fixed the bug. There is a need to devise a technique to distinguish between the committer and the actual fixer.

5.4. Reopened Bug Analysis

In some cases, reopened bug is considered as a new bug and tossed before assigning to the developer. Although the active developer who last resolved this bug might be able to resolve the issues related to reopening of the bug. Hence a different triaging approach needs to be adopted for reopened bugs.

5.5. Load Balancing

In many Triagers, a newly reported bug is only assigned to experienced and relevant developers (D) without considering their workload. This may slow down the process of bug fixing. In this situation, Load balancing reassignment can be beneficial, and bugs might get fixed faster.

6. Conclusion

In our survey paper, we studied various techniques that are based on machine-learning approach, Metadata based approach and Profile based approach. We have noticed that none of the approach is up to mark in term of accuracy, performance and effectiveness, like: Machine learning approaches worked on previously resolved bug reports for assigning a bug to relevant developer. Sometimes an existing developer may not have enough knowledge for bug fixing and cannot be consider a new developer as a relevant bug fixer. Although Metadata based approaches considered all type of bug data like: developer identifier, time-stamp, commit, and comments etc for fixing the new bug, but still face a problem to select a relevant developer for a purely new bug which has different contents from the previously fixed bugs. According to developer specialization, a developer rank-list is maintained in profile based approaches and this approach is capable to reduce developer sparseness also. We have notice that these approaches are not consider any of the developer rank metrics, developer raking process also have an unclear way in bug triaging and all the bugs are not equally distributed between the relevant developer i.e. some of the developers are heavily loaded. These couple of issues is affected the bug fixing process. This paper has brought in light several important issues related to load balancing, cold- start, developer activity etc.

Acknowledgments

This work is supported by JIIT, Noida and authors would also like to thanks the anonymous reviewer for his/her detailed comments.

References

[1]. Jaweria Kanwal , Onaiza Maqbool“ Bug prioritization to facilitate Bug report Triage” *Journal of Computer Science and Technology* , March 2012, Volume 27, Issue 2, pp 397-412

[2]. Mamdouh Alenezi, Kenneth Magel, Shadi Banitaan “Efficient Bug Triage using text Mining” 2013 *ACADEMY PUBLISHER* doi:10.4304/jsw.8.9.2185-2190

[3]. Lalita Sharma, Anju Gera “A survey on recommendation system: Research Challenges” *International Journal of Engineering Trends and Technology (IJETT)* - Volume4Issue5- May 2013

[4]. Ahmed E. Hassan “The Road Ahead of Mining Software Repository” Frontiers of Software Maintenance, Beijing, Publisher*: IEEE*, Sept. 28 2008-Oct. 4 2008

[5]. Awoosung JUNG, Eunjoo Lee and Chisu Wu: “A survey on Mining Software repositories”,*ICICE trans.Inf & Syst*,vol-95, 5may 2012,

[6]. Tamrawi A, Nguyen T T, Al-Kofahi J, et al. “*Fuzzy set-based automatic bug triaging”.* In: Proceedings of the International Conference on Software Engineering, Waikiki, 2011. 884–887

[7]. D. Cubranic, “*Automatic bug triage using text categorization”* In In SEKE 2004: Proceedings of the Sixteenth International Confrence on Software Engineering & Knowledge Engineering, Citeseer, 2004

[8]. Huzefa Kagdi, Malcom Gethers, Denys Poshyvanyk, Maen Hammad“Assigning change requests to software developers” *Journal of Software: Evolution and Process*, Volume 24, Issue 1, pages 3–33, January 2012

[9]. Jonas Helming, Holger Arndt, Zardosht Hodaie , Maximilian Koegel , Nitesh Narayan “Automatic assignment of work item” *Evaluation of Novel Approaches to Software Engineering* Volume 230 of the series Communications in Computer and Information Science pp 236-250

[10]. John Anvik, Gail C. Murphy, “Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions”, *ACM Transactions on Software Engineering and Methodology*, Volume 20 Issue 3, August 2011 Article No. 10

[11]. Pamela Bhattacharyaa , Iulian Neamtiua , Christian R. SheltonaAutomated, “ Highly-accurate Bug Triage using Machine Learning*” Journal of Systems and Software* November 29, 2010

[12]. Gaeul Leong, Sunghun Kim “ *Improving bug triage with Bug tossing Graphs*” ESEC/FSE ’09 proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering pages 111-120

[13]. Pamela Bhattacharya, Iulian Neamtiu, Christian R. Shelton , “Automated, highly-accurate, bug assignment using machine learning and tossing graphs”, *Journal of System and Software*, Volume 85 Issue 10, October, 2012 Pages 2275-2292

[14]. Liguo Chen, Xiaobo Wang, Chao Li “An Approach to Improving Bug Assignment with bug tossing graph and bug similarities” *Journal of software*, vol 6 No3 (2011), 421-427, Mar 2011

[15]. V. Akila , G. Zayaraz “Novel metrics for bug triage” *journal of software*, vol 9, No 12 (2014), pags: 3035. 3040

[16]. Jifeng Xuan , He Jiang “*Automatic Bug Triage using Semi-Supervised Text Classification*”, Proc. 22th Intl. Conf. Software Engineering & Knowledge Engineering (SEKE ’10)

[17]. Jin-woo Park, Mu-Woong Lee, Jinhan Kim, Seung-won Hwang, Sunghun Kim “COSTRIAGE: A Cost-Aware Triage Algorithm for Bug Reporting”, *AAAI, AAAI Press*, (2011)

[18]. Ramin Shokripour, John Anvik, Zarinah M. Kasirun, Sima Zamani “A time based approach to Automatic Bug Report Assignment”, *Journal of Systems and Software* , Volume 102, April 2015, Pages 109–122

[19]. Tung Thanh Nguyen, Anh Tuan Nguyen, Tien N. Nguyen “Topic- based, time aware bug assignment” *ACM SIGSOFT Software Engineering Notes,* Volume 39 Issue 1, January 2014 , page 1-4

[20]. Ramin Shokripour, John Anvik, Zarinah M. Kasirun, Sima Zamani “Improving automatic bug assignment using time- meta in term weights”, *Software, IET* , Volume:8 Issue:6

[21]. Hao Hu, Hongyu Zhang ; Jifeng Xuan ; Weigang Sun “*Effective Bug Triage based on Historical Bug-Fix information”*, Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on 3-6 Nov. 2014

[22]. Ramin Shokripour, Zarinah M. Kasirun, Sima Zamani, John Anvik “*Automatic Bug Assignment Using Information Extraction Methods”* Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on 26-28 Nov. 2012

[23]. Sima Zamani , Sai Peck Lee, Ramin Shokripour, John Anvik “ A Noun based approach to feature location using time aware term- weighting” *Information and Software Technology*, Volume 56, Issue 8, August 2014, Pages 991–1011

[24]. Tao Zhang, Byungjeong Lee *“An Automated Bug Triage Approach: A Concept Profile and Social network Based Developer Recommendation*” 8th International Conference, ICIC 2012, Huangshan, China, July 25-29, 2012. Proceedings

[25]. Hoda Naguib, Nitesh Narayan, Bernd Brugge, Dina Helal *"Bug report assignee Recommendation using Activity Profile*" 2013 10th Working Conference on Mining Software Repositories (MSR) pp: 22-30

[26]. Tao Zhang, Byungieong Lee, *"A Hybrid Bug Triage Algorithm for Develop recommendation*" Proceeding SAC '13 Proceedings of the 28th Annual ACM Symposium on Applied Computing, pages 1088-1094

[27]. Mamdouh Alenezi, Kenneth Magel, Shadi Banitaan "Efficient Bug Triaging Using Text Mining" *Journal of Software*, Vol 8, No 9 (2013), 2185-2190, Sep 2013

[28]. Jifeng Xuan, Yan Hu, He Jiang "Dept- Prone Bugs: Technical Dept in Software Maintenance" *International Journal of Advanced in Computing Technology (IJACT)*, Volume4, Number19, October. 2012

[29]. Jifeng Xuan, He Jiang , Yan Hu , Zhilei Ren "Towards effective Bug Triage with Software data reduction techniques", *Knowledge and Data Engineering, IEEE Transactions* on (Volume:27 , Issue: 1)

[30]. Sarah Rastkar, Gail C. Murphy , Gabriel Murray " Automatic summarization of bug reports*", IEEE Transactions on Software Engineering* Volume 40 Issue 4, April 2014 Page 366-380

[31]. Cubrani´c D. *Automatic bug triage using text categorization*. In: Proceedings of the International Conference on Software Engineering & Knowledge Engineering, Alberta, 2004. 92–97

[32]. Cavalcanti, Y.C., Neto, P.A.d.M.S., Machado, I.D.C., Vale, T.F., de Almeida, F.S., de lemos, Meira, S.R., 2014, "Challenges and opportunities for software change request repositories; a systematic mapping study". *J. Softw.Evol. Process* 26(7) 620-653

[33]. B. Sisman, A.C. Kak, *Incorporating version histories in information retrieval based bug localization*, in: 9th IEEE Working Conference on Mining Software Repositories (MSR), IEEE, 2012, pp. 50–59

[34]. Ossama H. Embarak "*A Method for Solving the cold start problem in recommendation system*", International conference on Innovations in information Technoligy, 2011

[35]. Alenezi, M, Banitaan, S. "*Bug Reports Prioritization: Which Features and Classifier to Use?"* Machine Learning and Applications (ICMLA), 2013 12th International Conference on (Volume:2)