Journal of Software Engineering and Applications, **2012, 5, 255-261**
http://dx.doi.org/10.4236/jsea.2012.54030 Published Online April 2012 (http://www.SciRP.org/journal/jsea)

Rank-Me: A Java Tool for Ranking Team Members in Software Bug Repositories

Naresh Kumar Nagwani[1], Shrish Verma[2]

[1]Department of Computer Science & Engineering, National Institute of Technology Raipur, Raipur, India; [2]Department of E&TC, National Institute of Technology Raipur, Raipur, India.
Email: {nknagwani.cs, shrishverma}@nitrr.ac.in

Received January 21st, 2012; revised February 29th, 2012; accepted March 25th, 2012

ABSTRACT

In this paper a team member ranking technique is presented for software bug repositories. Member ranking is performed using numbers of attributes available in software bug repositories, and a ranked list of developers is generated who are participating in development of software project. This ranking is generated from the contribution made by the individual developers in terms of bugs fixed, severity and priority of bugs, reporting newer problems and comments made by the developers. The top ranked developers are the best contributors for the software projects. The proposed algorithm can also be used for classifying and rating the software bugs using the ratings of members participating in the software bug repository.

Keywords: Software Bug Repository; Team Member Rating; Rating Software Bugs; Team Member Scores in Bug Repositories

1. Introduction

A software bug repository contains information about the software bugs. A software bug report consist of some bug attributes like summary (or title) of the bug, description of the bug, date at which the bug is reported, assigned-to field (developer to whom the bug has been assigned, reported-by field (person by whom the bug has assigned), comments from the team members etc. Large scale projects maintain their bug information's using bug tracking tools. Some of the popular bugs tracking tools are Bugzilla, Trac, JIRA, Perforce etc. An example of software bug repository using these tools is Mozilla Bugzilla, which has the information about the Mozilla project bugs. The present work reports a new algorithm to rate (rank) the team members on the basis of information available in bug repositories about their contribution.

1.1. Motivation

The motivation behind this work is to answer the following three questions: 1. Why to rate team members? 2. How to rate team members? 3. How this ranking will help? The answers to these questions give the motivation towards the proposed work, and are given in the following sections.

1.2. Why to Rate Team Members?

Rating team members in software bug repository helps in identifying the expert and good team members involved in software project. It is a measure by which various team members can be classified by the amount of effort each team member exerts in the project.

1.3. How to Rate?

The rating can be done using the vast information present in the software bug repository. The effort could be made in verity of ways; some of which are resolving or fixing the bugs for the different priority and severity levels. Person who has fixed critical severity and/or high priority bugs will get more score than one who has fixed low priority and/or minor severity bugs. Another effort could be on the basis of the number of times the user is providing the comments for the reported bugs to offer more information, and the number of times a team member is reporting new issues and bugs related to the software. Apart from this if the team member is tracking some bugs and is a part of bug thread communication (Carbon Copy mailing list), then also the effort needs to be calculated. Using all these efforts the scores for team member is calculated for their rating.

1.4. How This Ranking Will Help?

Rating the team members help in finding the experienced and efficient team members who can carry more responsibilities and can help in faster development of the soft-

ware. The good rating persons show that they are having good experience in the software project and they can guide junior or new team members for effective bug resolutions and faster development of the software.

2. Related Work

Bug repositories are huge source of informations. Using bug mining, knowledge patterns can be generated. Weib, *et al*. reported a method to predict the software bug estimation [1,2], proposed predictors are designed using two data points—the title, and description of bugs. An approach is designed by Matter *et al*. [3] to automatically suggest developers who have the appropriate expertise for handling a bug report. The developer expertise model is designed using the vocabulary found in their source code contributions which is compared with the vocabulary of bug reports. An advantage of this approach is the record of previous bug reports is not required; the proposed technique is also able to recommend developers who are not involved in fixing the bugs earlier. Canfora and Cerulo [4] have developed a technique to identify the most appropriate developers to resolve a new change request and also to predict the set of impacted source files for this new change request.

Anvik *et al*. [5] proposed an approach to automate bug assignment using machine learning techniques, in this proposed technique, when a new report arrives, the classifier produced by the machine learning technique suggests a small number of developers suitable to resolve the report. The only problem with this technique is it has low precision and recall values for the suggested developers. Anvik and Murphy [6] did an empirical evaluation of two approaches to locate expertise. The two approaches are based on mining the source and bug repositories. In the first approach, source code check-in logs are examined for the modules that contain the fixed source files. In the second approach, the bug reports from bug repositories are analyzed to locate expertise. Panjer [7] has explored the feasibility of using data mining tools to predict the time to fix a bug based on basic information known at the beginning of a bug's lifetime.

Nagwani and Verma proposed the prediction technique for the bug fix estimations [8], in which the estimation was perofmed using average fix time for the similar software bugs in the proposed technique. Nagwani and Verma [9] designed an open framework in java to pre-process the data available at online software bug repositories. The proposed framework accepts a URL (Uniform Resource Locator) for the online software bug repository, where user can specify the ranges of the software bug id's. The bugs in the specified ranges are stored to the local machine where the framework provides the parsing and pre processing work for the retrieved software bugs. The processed software bugs can be taken directly from the local machine for the analysis. A framework for automated assignment of bug-fixing tasks is presented by Baysal *et al*. [10], in the proposed approach the knowledge about a developer's expertise is used by analyzing the history of bugs previously resolved by the developer, in the proposed approach lacks the experimental evaluation due to number of reasons mentioned in the work.

A detailed literature review reveals that although the ranking mechanism is common at most of the places like applying the ranking in web search result [11] etc., but the concept of ranking the team members in software bug repositories is a novel technique and is yet to be reported. Gousios *et al*. [12] proposed a model by combining traditional contribution metrics with data mined from software repositories for developer contribution measurements. In the proposed model clusters of similar projects were created to extract weights which were applied to the actions performed by a developer on project assets to calculate a combined measurement of the developer's contribution. The proposed model was designed for overall contribution of developers, but the problem with the proposed model is, the number of parameters is calculated using personal experience and traditional contribution, which may not be feasible or accurate at certain cases, whereas in the present work, the member ranking is performed for software bug repositories using information available in software bug repository, itself.

3. Methodology

The overall process of assigning ratings to the team members in software bug repository is shown in **Figure 1**. Using the information present in bug attributes of the bug reports, various scores are calculated for each team member. The major scores are bug fixing score using importance (priority and severity) of bugs, user comment scores, newly reported bug scores and CC mailing list scores for bug related communication in software bug repository. The aggregate scores are calculated using the four scores, and normalized for the ratings. This criterion is used to develop the rating of the team members present in software bug repository.

The overall score of i'th team member in the software bug repository can be calculated using (1). This score is the linear sum of four major scores. These four scores are bug fixing score ($Score_i(Fix)$) according to the priority and severity of the software bugs, the score of the comments ($Score_i(C)$) made by the team members in various bugs, the score of the team members who are added in the CC mailing list ($Score_i(CC)$) in the various software bugs and the last score is the score of number of times a team member has reported a software bug ($Score_i(R)$).

$$\begin{aligned} Score_i &= Score_i(Fix) + Score_i(C) \\ &\quad + Score_i(CC) + Score_i(R) \end{aligned} \tag{1}$$

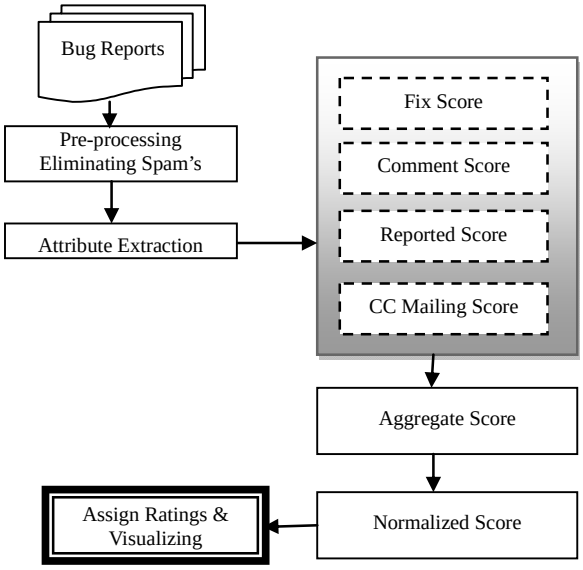

Figure 1. The rating methodology for the team members.

$Score_i(Fix)$ can be again decomposed in two the two parts as given in (2).

$$Score_i(Fix) = Score_i(P) + Score_i(S) \quad (2)$$

$Score_i(P)$ is bug fixing scores using different priority values and $Score_i(S)$ is the bug fixing scores using different severity values. The calculation of $Score_i(P)$ is based on assumption, that there are five different priorities available in software bug repositories named P_1, P_2, P_3, P_4, and P_5 (Most of the bigger projects like Mozilla, MySql, JBoss etc. have five priority levels. Each of the five different priorities are assigned corresponding weights, as given in (3). The relationship between the various priorities weighting factor is given) in (4).

$$\begin{aligned} Score_i(P) = {} & WP_1 * N_i(P_1) + WP_2 * N_i(P_2) \\ & + WP_3 * N_i(P_3) + WP_4 * N_i(P_4) \\ & + WP_5 * N_i(P_5) \end{aligned} \quad (3)$$

$$WP_1 > WP_2 > WP_3 > WP_4 > WP_5 \quad (4)$$

In the similar manner the severity score is also calculated using different weighting factor of different severities at different levels and is given in (5). The standard severity levels are typically the "Critical", "Major", "Normal", "Enhancement", "Trivial" and "Minor". For these different severities different score-weights are assigned to fix or resolve a bug. The relationship between the various severity score is presented in (6). The symbol used in (3) and (5) for calculating the priority and severity scores are systematically arranged and summarized in **Tables 1** and **2**.

$$\begin{aligned} Score_i(S) = {} & CT * N_i(CT) + MJ * N_i(MJ) \\ & + NM * N_i(NM) + EN * N_i(EN) \\ & + TR * N_i(TR) + MN * N_i(MN) \end{aligned} \quad (5)$$

Table 1. Various symbols used in equations.

Symbol	Significance
$N_i(P_1)$	Number of priority P_1 bugs fixed by a developer.
$N_i(P_2)$	Number of priority P_2 bugs fixed by a developer.
$N_i(P_3)$	Number of priority P_3 bugs fixed by a developer.
$N_i(P_4)$	Number of priority P_4 bugs fixed by a developer.
$N_i(P_5)$	Number of priority P_5 bugs fixed by a developer.
$N_i(CT)$	Number of Critical severity bugs fixed by a developer.
$N_i(MJ)$	Number of Major severity bugs fixed by a developer.
$N_i(NM)$	Number of Normal severity bugs fixed by a developer.
$N_i(EN)$	Number of Enhancement bugs fixed by a developer.
$N_i(TR)$	Number of Trivial severity bugs fixed by a developer.
$N_i(MN)$	Number of Minor severity bugs fixed by a developer.
$N_i(C)$	Number of comments made by a team member.
$N_i(CC)$	Number of time a team member is added to the CC mailing list in the bugs.
$N_i(R)$	Number of times a team member has reported a bug.

Table 2. Various weighting parameters and their meaning.

Weighting Parameter	Significance
WP_1	Weight of bug fixed by a developer with the priority P_1
WP_2	Weight of bug fixed by a developer with the priority P_2
WP_3	Weight of bug fixed by a developer with the priority P_3
WP_4	Weight of bug fixed by a developer with the priority P_4
WP_5	Weight of bug fixed by a developer with the priority P_5
CT	Weight of bug fixed by a developer with the severity Critical
MJ	Weight of bug fixed by a developer with the severity Major
NM	Weight of bug fixed by a developer with the severity Normal
EN	Weight of bug fixed by a developer with the severity Enhancement
MN	Weight of bug fixed by a developer with the severity Minor
TR	Weight of bug fixed by a developer with the severity Trivial
R	Weight of bugs reported by a team member
C	Weight of comments made by a team member
CC	Weight of being the part of CC mailing list for a bug.

$$CT > MJ > NM > EN > TR > MN \quad (6)$$

The score calculations for the bug comments, cc mailing list and number of bugs reported by the team member are given in (7) to (9) respectively.

$$Score_i(C) = C * N_i(C) \quad (7)$$

$$Score_i(CC) = C * N_i(CC) \quad (8)$$

$$Score_i(R) = C * N_i(R) \quad (9)$$

where $N_i(C)$, $N_i(CC)$ and $N_i(R)$ are the number of times the i^{th} team member posted comment, number of time the i^{th} team member is added to the CC mailing list and number of time the i'th team member has reported a software bug. C, CC and R are the weigh for the scores for the comments, CC mailing list and newly reported software bugs.

The methodology is elaborated further in the following continuation.

4. Pre-Processing and Data Preparation for Rating

Software bugs are available as HTML or XML file formats in online software bug repositories, where the software bugs are managed using various bug tracking tools like Bugzilla, Trac, Jira, Perforce etc. One instance of a software bug taken from Mozilla bug repository (available at: https://bugzilla.mozilla.org) is shown in **Figure 2**. Mozilla uses the Bugzilla bug tracking tool to manage the software bugs of Mozilla products. These bugs are required to be available in local machine in order to perform some analysis. Parsing is another major operation required in order to extract the various bug attributes information. In the present paper the framework proposed by Nagwani and Verma [10] is used for this purpose. Once the parsing is done, suitable attributes for team ranking are filtered from the parsed data. The attributes selected in this work are assigned-to, priority, severity, cc-list, and reported-by.

Apart from parsing, another important pre-processing operation is identification and elimination of spams from the various textual bug fields like comments, description etc. This will ensure that only the effective information is selected for the analysis. For instance, just to increase the comment count any user can put some ineffective information in comment section of any bug, which should not be counted for ranking of commented user since it is useless. For this purpose jASEN (java Anti Spam ENgine), an open source java based anti spam framework is used. The jASEN framework is available at http://www.jasen.org/

5. Score Transformation and Rating Normalization

Once the raw score for the team members participating in the software bug repository is calculated, the next important step is to transform the raw score into the newer range of score for smoothing the criteria of ratings. The

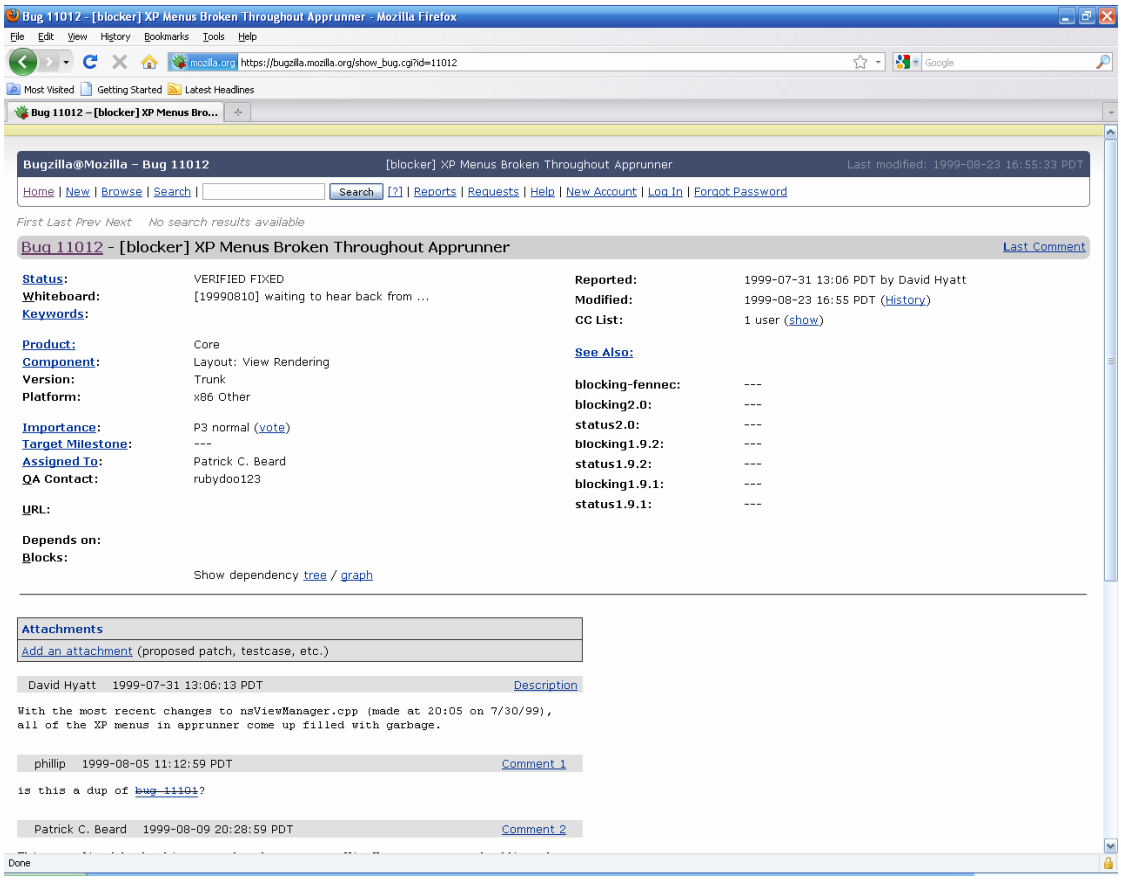

Figure 2. Snapshot of a Mozilla bug (bug id = 11012).

reason behind normalization of raw score is to make the rating outputs was to understand. For this purpose various score normalization techniques are available. Out of which the most popular is Min-Max normalization, as explained below.

$$n' = (n - \min_A) * (\text{Max}_{NEW} - \text{Min}_{NEW}) / (\max_A - \min_A) + \text{Min}_{NEW} \quad (10)$$

where ν' is the transformed value for the given value ν for a particular attribute A, $\max_A$ and $\min_A$ are the maximum and minimum values of an attribute A. Max_{NEW} and Min_{NEW} are the new maximum and new minimum values; the new range where the given value is going to be transformed. The attribute "A" here is the score calculated for participating team members in a software bug repository.

Alternatively, normalization using decimal scaling and other standard normalization techniques can also be used. However in the example presented in this paper Min-Max normalization technique is used with minimum value as 0 and maximum value as 5.

6. Visualizing the Member Ranking

Next the team member ratings have to be visualized using the ranking value calculated from the proposed algorithm. Two common rating visualization techniques are mentioned here as a reference. **Figure 3** shows a star rating visualization to represent the rating in the range of [0…5], Where as **Figure 4** represents the rating bar visualization for rating scores in the scale of 0 to 5.

7. Calculating Bug Ranking Using Member Ranking

The proposed team member ranking can now be used for calculating software bug ranks. The software bug rank can be used to represent the ordering of the bugs using the involvement of team members of different rank. Linear sum of rank of the team members involved in software bugs can used for calculating the software bug rank. A general way of calculating the software bug rank is given in (11).

$$\text{BRank} = (\text{Assigned}-\text{by})_{RANK} + (\text{Assigned}-\text{To})_{RANK} + (\text{Comment}_1)_{RANK} + (\text{Comment}_2)_{RANK} + \cdots + (\text{Comment}_N)_{RANK} \quad (11)$$

After calculating the bug rank it can be normalized to a specific range to indicate various bug rank levels.

8. Implementation & Experiment

The implementation is performed for the given technique

Figure 3. A star rating visualization technique in the range [1…5].

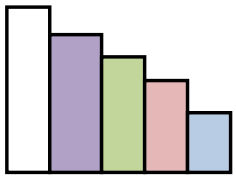

Figure 4. A rating bar visualization technique in the range [1…5].

using Java as the programming language and MySql as the local database to store the processed information. The implementation is performed in the five stages.

1) Retrieving and Parsing the Software Bugs: In the first step the software bugs are retrieved at local system and parsing using tokenization for extracting the bug attributes and their corresponding values.

2) Creating local database for selected attributes: The extracted attributes are filtered for analysis and saved in the local database.

3) Eliminating the possible spams: In this stage the textual bug attributes are analyzed for possible spam. The information which is likely to be spam is ignored and never used for calculating the ranking of team members.

4) Generating Metadata for Ranking: Once the possible spams are eliminated the next step is to prepare the metadata for ranking. This includes counting the number of team members; number of bugs for each member, number of comments for each member etc. is performed in this stage.

5) Implementing the ranking algorithm: With the help of metadata generated in previous stage and using various user supplied weights given in **Table 2**, the algorithm given in section-2 is implemented in java.

6) Visualizing the ranking: As a post processing part after the calculation of ranking for the team members, using one of the visualization techniques can be used to visualize the normalized rank of a team member.

For experiment and simulation of the developed algorithm Mozilla bug repository is selected and around 1000 software bugs are selected using sequence sampling technique. These bugs are selected from bug-id 501 to 1500. The list of developers is generated, total 97 developers were found to whom these bugs were assigned. For each developer total numbers of bugs are counted, which indicates the involvement of a developer (or team member) in the process of bug resolution. The experiment for team member ranking demonstration is performed using the weight values of R = 5, C = 2, CC = 1, WP_1 = 10, WP_2 = 8, WP_3 = 6, WP_4 = 4, WP_5 = 2, CT = 10, MJ = 8, NM = 6, EN = 4, and MN = TR = 2.

The output data about the top five contributors is computed and is shown in **Table 3** for the selected 1000 sample (bug-id 501 to 1500) bugs from Mozilla bug repository. In this table the computation details of members are provided, who have contributed more in the software bug repository for the selected 1000 samples. For example "BUSTER" is one of the member whose bug fix score is 2796 (which is calculated using Equation (2) by combing the various bug fix scores for different priority and severity software bug), bug reported score is 0 (bug reported score is 0, since non of the selected bug samples are reported by "BUSTER"), comment score is 296, CC-listing score is 61, total score is 3449 and the transformed score is 5.0 since "BUSTER" got the maximum total score in the selected samples.

The fix score calculation break up for the team members is given in **Table 4**. For example "BUSTER" have worked on fixing 22 priority "P1" bugs etc., using these counts weighted sum is calculated for the different priority and severity bugs using the appropriate weights explained in **Table 2**. The totcal scores of the team members are calculated using Equation (2). The normalized marks are also shown in the last column of the table after the Min-Max normalization with Min_{NEW} as 0 and Ma_{NEW} as 5.

Table 6 shows the breakup of the bug fix score using the different priorities and severities of the software bugs. The bug fix value of the team member 'LEGER' is 0 which means he was not involved in bug fixing rather he was contributing by providing the "757" comments in the selected bug samples, which made him the top contributor. The sample rating criteria is shown in **Table 5** (selected for the experiment) and the final output in form of the ratings of the top 5 contributors using the sample rating criteria is given in **Table 6**. The final output shows that the rating of only user

"BUSTER" is outstanding, and he is found to be very effective person for the 1000 sample software bugs.

9. Limitation of Proposed Work

Literature review reveals that the ranking technique like the proposed one is not yet implemented. Once this type of approach will be introduced for the software repositories, there is the possibility that some of the team members can start posting of ineffective information in such repositories. So for effective ranking of team member the management (project managers) has to monitor the effectiveness of the team members. Various weights given in **Table 2** can be adjusted according to the manual monitoring in the bug repositories.

10. Conclusion & Future Scpope

A new algorithm to rate the participating team member in

Table 3. The top 5 scores from 1000 sample Mozilla bug reports.

Member	Fix	Reported	Comment	CCListed	Total Score	t-Score
BUSTER	2796	0	296	61	3449	5.0
LEGER	0	0	757	18	1532	2.22
KARNAZE	974	0	177	0	1328	1.92
PETER LINSS	834	0	116	0	1066	1.54
RICKG	794	0	112	34	1052	1.52

Table 4. Bug fix scores for the the top 5 scorers in 1000 sample Mozilla bug reports.

Member	P1	P2	P3	P4	P5	Critical	Major	Normal	Enhancement	Minor	Fix Score
BUSTER	22	156	13	3	1	11	27	148	2	7	2796
LEGER	0	0	0	0	0	0	0	0	0	0	0
KARNAZE	9	54	4	0	0	4	9	52	0	2	974
PETER LINSS	3	51	5	0	0	3	2	52	2	0	834
RICKG	10	42	1	0	0	6	8	37	1	1	794

Table 5. Sample criteria for team members rating.

Criteria-id	Criteria	Rating
1	t-Score > 4.0	Outstanding
2	t-Score <= 4.0 && t-Score > 3.0	Excellent
3	t-Score <= 3.0 && t-Score > 2.0	Very Good
4	t-Score <= 2.0 && t-Score > 2.0	Good
5	t-Score <= 1.0 && t-Score > 0.0	Moderate

Table 6. Ratings for top scorers in 1000 sample bug reports in Mozilla.

Member	Score	t-Score	Rating
BUSTER	3449	5.0	Outstanding
LEGER	1532	2.22	Very Good
KARNAZE	1328	1.92	Good
PETER LINSS	1066	1.54	Good
RICKG	1052	1.52	Good

a software bug repository is presented in this paper. Various scores required to rate a team member are formulated and calculated by assigning proper weights to each score. Experiments are performed and validated over Mozilla bug repository, and the calculated ratings are validated and found to be in good agreement. Future scope of this work can be identifying the relevance of user comments and assigning the weight as per the relevance of the comment entered by the team member. Also the technique of team member's contribution in terms of effectiveness can be evolved.

REFERENCES

[1] C. Weiss, R. Premraj, T. Zimmermann and A. Zeller, "How Long Will It Take to Fix This Bug?" *Proceedings of the 4th International Workshop on Mining Software Repositories*, Minneapolis, 19-20 May 2007, pp. 1-8. doi:10.1109/MSR.2007.13

[2] C. Weiss, R. Premraj, T. Zimmermann and A. Zeller, "Predicting Effort to Fix Software Bugs," *Proceedings of 9th Workshop Software Reengineering* (*WSR* 2007), Bad Honnef, May 2007.

[3] D. Matter, A. Kuhn and O. Nierstrasz, "Assigning Bug Reports Using a Vocabulary-Based Expertise Model of Developers," *Proceedings of 6th IEEE Working Conference on Mining Software Repositories* (*MSR* 2009), Vancouver, 16-17 May 2009, pp. 131-140. doi:10.1109/MSR.2009.5069491

[4] G. Canfora and L. Cerulo, "How Software Repositories can Help in Resolving a New Change Request," *Proceedings of IEEE International Workshop on Software Technology and Engineering Practice* (*STEP* 2005), Budapest, September 2005.

[5] J. Anvik, L. Hiew and G. C. Murphy, "Who Should Fix This Bug?" *Proceedings of the 28th International Conference on Software Engineering* (*ICSE* 2006), Shanghai, 20-28 May 2006, pp. 329-338. doi:10.1145/1134285.1134336

[6] J. Anvik and G. C. Murphy, "Determining Implementation Expertise from Bug Reports," *Proceedings of 4th International Workshop on Mining Software Repositories* (*MSR* 2007), Minneapolis, 20-26 May 2007, p. 2. doi:10.1109/MSR.2007.7

[7] L. D. Panjer, "Predicting Eclipse Bug Lifetimes," *Proceedings of 29th International Conference on Software Engineering Workshops* (*ICSEW* 2007), Minneapolis, 20-26 May 2007, p. 29. doi:10.1109/MSR.2007.25

[8] N. K. Nagwani and S. Verma, "Predictive Data Mining Model for Software Bug Estimation Using Average Weighted Similarity," *Proceedings of IEEE 2nd International Advance Computing Conference* (*IACC* 2010), Patiala, 19-20 February 2010, pp. 373-378. doi:10.1109/IADCC.2010.5422923

[9] N. K. Nagwani and S. Verma, "An Open Source Framework for Data Pre-Processing of Online Software Bug Repositories," *CiiT International Journal of Data Mining Knowledge Engineering*, Vol. 1, No. 7, 2009, pp. 329-338.

[10] O. Baysal, M. W. Godfrey and R. Cohen, "A Bug You Like: A Framework for Automated Assignment of Bugs," *Proceedings of 17th International Conference on Program Comprehension* (*ICPC* 2009), Vancouver, 17-19 May 2009, pp. 297-298. doi:10.1109/ICPC.2009.5090066

[11] L. Page, S. Brin, R. Motwani and T. Winograd, "The Pagerank Citation Ranking: Bringing Order to the Web," *Technical Report*, Stanford University, Stanford, 1998.

[12] G. Gousios, E. Kalliamvakou and D. Spinellis, "Measuring Developer Contribution from Software Repository Data," *Proceedings of the 28th International Working Conference on Mining Software Repositories* (*MSR* 2008), Leipzig, 10-11 May 2008, pp. 129-132. doi:10.1145/1370750.1370781

