International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 11 (2017) pp. 2840-2847

Reduction of Duplicate Bugs and Classification of Bugs using a Text Mining and Contingency Approach

Rajeshwari M R
Department of Computer Science,
Ghousia College of Engineering,
Ramanagara, Karnataka, India.

Dr. Kavitha K S
Department of Computer Science,
Global Institute of Technology,
Bangalore, Karnataka, India.

Abstract
The numbers of software products coming out day by day are increasing in the market. Due to the growing demands of the community and to satisfy consumer needs in various domains. In Software Development Lifecycle the steps starts from the collection of requirements, discussion of estimates, allocation of resources, design, development, quality assurance. It so happens that due to high importance for time to market the quality is degraded. The Quality assurance team has to deal with huge number of bugs within a specific timeline and has to triage the bugs and make the bugs which are in common into a group so that the effort is reduced.
In this paper first we have a predefined set of bugs across 4 different products. The bugs undergo a series of data mining steps or algorithms namely cleaning, frequency computation, IDFT computation, score computation and then the duplicate bug detection algorithm is applied which groups the bug. The second part of the project is responsible for classification of the bugs in terms of user interface, design, performance and usability. The classification is performed by applying text based naive classifier which finds the probability, contingency and enhanced contingency on the bugs and then assigns a class for the bug.

Keywords: Bug triage, tokenization, IDFT.

BACKGROUND

In the paper [1] the authors propose that open source development projects through the triage and priority must be assigned on a manual basis. The manual process is avoided by the approach used in the algorithm proposed in the context.
In the paper [2] it is described that there are common errors in web pages are script crashes and malformed web pages. The paper also describes an approach to get automatically generated test cases for a web page. In the paper [3] the authors describe an algorithm that presents a machine learning approach to create a bug repository and then recommendations are created based on group of bugs. The algorithm also goes future to propose developers who can fix the bugs quickly.
In the paper [4] the authors propose a vector space model for the huge text data representation. The disadvantage of vector space model is that it does not maintain the ordering of the words therefore authors produces an approach in which distance between the words in the graphs are used to intercept the information in terms of sentence structure of the underlying data.

PROPOSED FRAMEWORK

In the current approach first a list of bug reports across streams namely Eclipse, Mozilla and Open office are collected and then they are cleaned using a stream of stop words. Once the clean data is obtained the tokenization is performed on the bug reports and text frequency is compute. After performing IDFT computation, Feature Vector is computed and list of categorical similarities and cosine similarity are measured.

METHODOLOGY

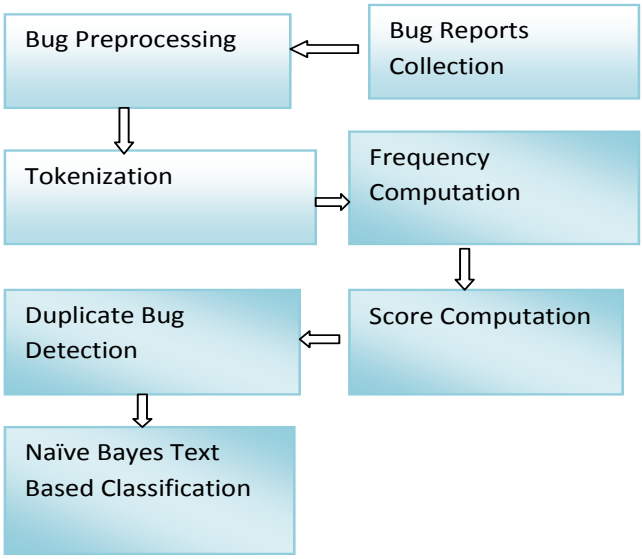


Figure- 1: Methodology

A. Collection of Bug Reports

The bugs for all the above products namely Software Engineering for any of product, Mozilla, open office and eclipse. All the bugs will be collected as a set {Bug Id, Component, Priority, Type, Version, Status, Description}.

B. Bug Preprocessing

This module is used in order to remove stop words from the bug description. The stop words used in this project are standard words given in the web mining forums. The stop words are namely *able, about, above, abroad, according, accordingly, across, actually, adj, after, afterwards, again, against, Ago, ahead, ain't, all, allow, allows, almost, alone, along, alongside, already, also, although, always, am, Amid, amidst, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, a's, aside.*

C. Tokenization

Tokenization is a process of converting the clean bug into a sequence of tokens. Each token is associated with a bug {TokenId, TokenName, BugId, ProductId}.

D. Frequency Computation

It is defined as the number of times a token appears in the review. The frequency will remove if any redundancy exists. The frequency is stored in the format {FreqId, TokenName, Freq ,BugId, ProductId}.

E. Score Computations

The Score computation is performed per token and is computed across the bugs by using the below formula and is stored in the format {ScoreId, Frequency, IDFT, Score, BugId, ProductIs}.

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D).(k_1+1)}{f(q_i, D) + k_1 (1 - b + b).\frac{|D|}{avgdl}}$$

$f = frequency$

$IDF = Inverse\ Document\ Frequency$

$D = length\ of\ document$

$avgdl =$ average document length in the text collection

$k1 = 1.2$

$b = 0.75\ IDF(qi)$

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

$N = number\ of\ documents$

$n(q_i) = number\ of\ documents\ containing\ q_i$

The textual similarity is computed by comparing 2 bugs with respect to title and description.

$$textual_{description}(d_1, d_2) = BM25F(d_1, d_2)$$

Where,

$d_1 = document1$

$d_2 = document2$

F. Duplicate Bug Detection

The algorithm is used to detect whether the 2 bugs are similar or not. The algorithm finds the intersection sum and union sum and then the bugs are found in terms of grouping.

1. Consider the two bugs to be compared
2. Find the List of Key Phrases in Bug A1
3. Find the List of Key Phrases in Bug A2
4. Find the intersection set between List of Key Phrases in Bugs A1 and List of Key Phrases in Bugs A2.
5. Find the union set between List of Key Phrases in Bugs A1 and List of Key Phrases in Bugs A2..
6. Start from index1 till the end of key phrases in the intersection set
   a) Obtain the kth key phrase K
   b) Measure the text frequency of Bugs A1 for K
   c) Measure the text frequency of Bugs A2 for K
   d) if tf(k,A1)>=tf(k,A2) measure the intersection sum as below

intersectionsum =intersectionsum+tf(k,A1)

else

intersectionsum = intersectionsum+tf(k,A2)

   e) k=k+1
   f) Repeat the process from step a to step e until all tokens in the intersection set is exhausted.

7.Start from index1 till the end of key phrases in the union set
   a) Obtain the kth key phrase K
   b) Measure the text frequency of Bugs A1 for K
   c) Measure the text frequency of Bugs A2 for K
   d) if tf(k,A1)<tf(k,A2) measure the intersection sum as below

unionsum = unionsum +tf(k,A1)

else

unionsum = unionsum +tf(k,A2)

   e) k=k+1
   f) Repeat the process from step a to step e until all tokens in the union set are exhausted
   g) Measure Similarity =IntersectionSum/UnionSum

G. Naive Bayes Classifier

1) Obtains the bugs from the collection.
2) For each of the bugs the probability is computed using the following formula.

$$P(b|C_i) = \frac{Number\ of\ words\ of\ category\ C_i}{Total\ Number\ of\ Words}$$

$1 \le C_i \le 4$

3) Also the negative probability is also computed for each of the bug.
4) The probability computation is computed and constructed as below.

PROBABILITY	BUGID	CATNAME	NEGATIVEPROBABILITY	COUNT	TOTALWORDS

BugID – ID of the Bug
Probability- Positive Probability
CatName – c1,c2,c3 and c4
NegativeProbability – Finding the negative probability
Count- Number of words for the category
TotalWords- Number of words

5) The contingency is measure using the following.

$$Total\ Positive\ Other_{c1} = p(C2) + p(c3) + p(c4)$$

$$Total\ Negative\ Other_{c1} = p^1(c2) + p^1(c3) + p^1(c4)$$

6) The enhanced contingency is measured using the following equation.

$$Positive\ Category\ Ratio_{c1} = p(c1) + Total\ Negative\ Other_{c1}$$

$$Other\ Category\ Ratio_{c1} = p^1(c1) + Total\ Positive\ Other_{c1}$$

7) The bugs are then classified by order by positive category ratio maximum and other category ratio minimum.
8) The count for each category bugs are then made.

EXPERIMENTAL RESULT

Web based software is used in which the developer first registers by giving his/her preferences and type of work. Two types of users are used namely Admin and Developers. The Admin is responsible for sequence of algorithm operations as described in the methodology where as developers receive a bug based on their expertise.

A. Login Page

Figure2: Login Page

Figure-2 shows the Login Page through which admin logs into the system and executes the algorithm.

B. Bug Collection View

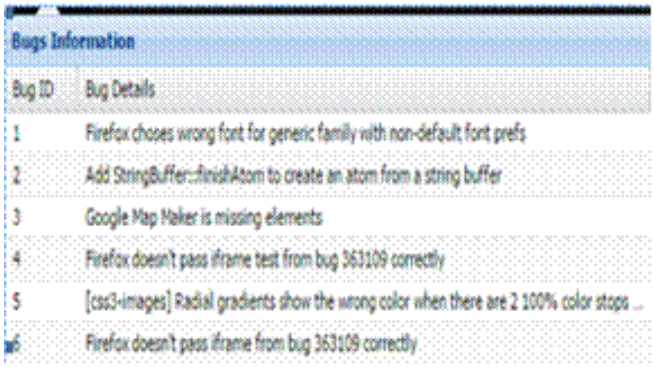


Figure 3: Bug Collection View

Figure-3 shows the list of bugs for the Firefox browser which admin can view after login.

C. Data Cleaning Algorithm Output

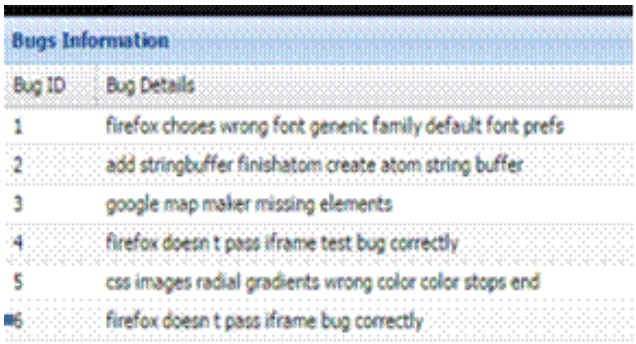


Figure 4: Data cleaning output

Figure-4 shows the data cleaning output. As shown in the result all stop words are removed from the bug details.

D. Tokenization and frequency Output

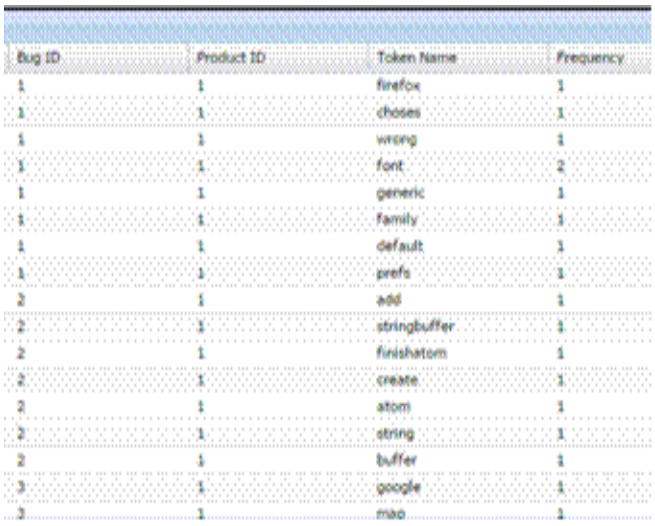


Figure 5: Frequency output

Figure-5 shows the Frequency output as show in the matrix unique tokens are shown and then frequency is also shown.

E. Score Computation Output

Token Name	IDF	NVALUE	Score
firefox	0	6	40.1387189782676
choses	0.564271430438563	6	30.2480906732828
wrong	0.255272505103306	6	33.2450443999954
font	0.564271430438563	6	34.4984981468701
generic	0.564271430438563	6	30.2480906732828
family	0.564271430438563	6	30.2480906732828
default	0.564271430438563	6	30.2480906732828
prefs	0.564271430438563	6	30.2480906732828
add	0.564271430438563	6	31.2093832134072
stringbuffer	0.564271430438563	6	31.2093832134072
finishatom	0.564271430438563	6	31.2093832134072
create	0.564271430438563	6	31.2093832134072
atom	0.564271430438563	6	31.2093832134072
string	0.564271430438563	6	31.2093832134072
buffer	0.564271430438563	6	31.2093832134072
google	0.564271430438563	6	33.3277106103457

Figure 6: Score Computation

Figure-6 shows the Score computation output which contains the value computed for Inverse Computation Frequency (IDF), N value and Score.

F. Score Computation Output Continued

Token Name	Average D	Small N	B Value	Document Magnitude
firefox	7.16666666666667	3	0	8
choses	7.16666666666667	1	0.423203572828922	8
wrong	7.16666666666667	2	0.19145437882748	8
font	7.16666666666667	1	0.423203572828922	8
generic	7.16666666666667	1	0.423203572828922	8
family	7.16666666666667	1	0.423203572828922	8
default	7.16666666666667	1	0.423203572828922	8
prefs	7.16666666666667	1	0.423203572828922	8
add	7.16666666666667	1	0.423203572828922	8
stringbuffer	7.16666666666667	1	0.423203572828922	7
finishatom	7.16666666666667	1	0.423203572828922	7
create	7.16666666666667	1	0.423203572828922	7
atom	7.16666666666667	1	0.423203572828922	7
string	7.16666666666667	1	0.423203572828922	7
buffer	7.16666666666667	1	0.423203572828922	7
google	7.16666666666667	1	0.423203572828922	7

Figure 7: Score Computation Output

Figure-7 shows the values of the score formula computed token wise namely Average D, Small N, B Value and Document Magnitude.

G. Duplicate Bug Detection

Main Bug ID	Bug ID	Group ID
1	1	1
1	2	0
1	3	0
1	4	0
1	5	0
1	6	0
2	2	2
2	3	0
2	4	0
2	5	0
2	6	0
3	3	3
3	4	0
3	5	0
3	6	0
4	4	4

Figure-8 : Duplicate Bug Detection

Figure-8 shows duplicate bug detection which has 3 values Main Bug Id, Bug Id and Group Id. After apply the algorithm many bugs will belong to same group.

Union Sum	Intersection Sum	Similarity
1	1	1
477.588397385397	0	0
425.761267943276	0	0
488.841071964802	40.1387189782676	0.0821099561396054
475.109757078114	33.2450443999954	0.0699733985773092
462.969215367913	40.1387189782676	0.0866984621134477

Figure 9: Duplicate Bug Parameters

Figure-9 shows the computation of duplicate bug parameters. As shown in the Fig-9 there is union sum, intersection sum and similarity if similarity is greater than threshold (0.6 or 0.7 or 0.8)

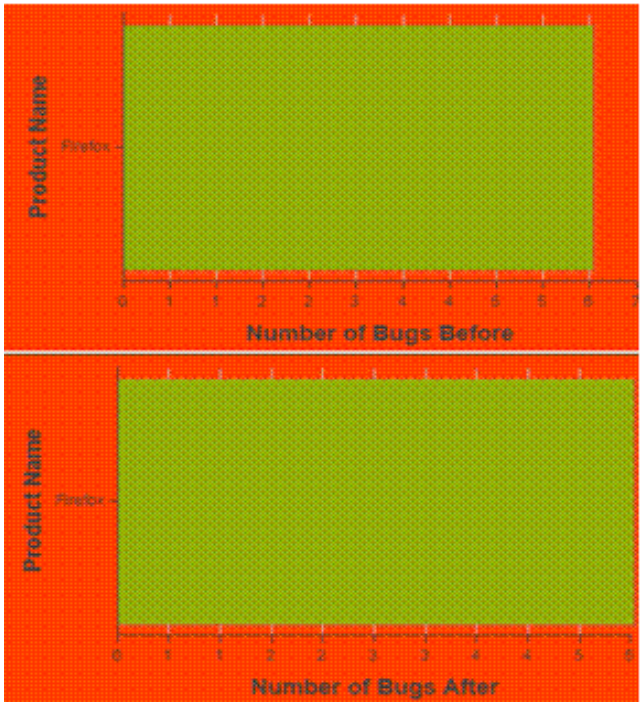


Figure 10: Comparison of Duplicate Bug Detection

Figure-10 shows the duplicate bugs. Before the number of bugs is 6 after that number of bugs are 5.

H. Classification Result

The classification results are described as below

Category Word	Category
quick response	Performance
extjs	Usability
javascript	Usability
jquery	User Interfacce
look and feel	Usability
database	Design
design	Design
load	Performance
resize	Usability
font	Usability
iframe	User Interface
Radial	Usability

Figure 11: Category words

Figure-11 shows the category words. As shown in the Fig-11 there is category word and category to which the word belongs.

I. Probability Computation

The probability computation results are shown in the tabular format.

Bug ID	Category Name	Count	Total Words	Probability	Negative Probability
1	Performance	0	9	0	1
1	Usability	2	9	0.22222222222	0.777777777777778
1	User Interfacce	0	9	0	1
1	Design	0	9	0	1
1	User Interface	0	9	0	1
2	Performance	0	7	0	1
2	Usability	0	7	0	1
2	User Interfacce	0	7	0	1
2	Design	0	7	0	1
2	User Interface	0	7	0	1

Figure 12: Probability Computation

Figure-12 shows the probability computation for the 2 bugs namely bug1 and bug2. The positive and negative probability for each category also has been computed.

J. Contingency

Contigency Information

Bug ID	Category Name	Negative Others	Positive Others
1	Performance	3	0
1	Usability	4	0
1	User Interfacce	3	0
1	Design	3	0
1	User Interface	3	0
2	Performance	4	0
2	Usability	4	0
2	User Interfacce	4	0
2	Design	4	0
2	User Interface	4	0

Figure 13: Contingency

Figure-13 shows the contingency output. The positive others are the positive of other category and negative others is the probable weight of other categories.

K. Enhanced Contingency

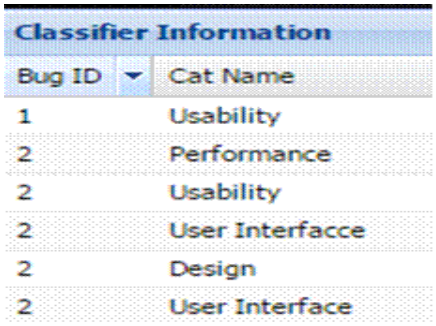
Enhance Matrix

Bug ID	Category Name	Positive Cat Ratio	Others Cat Ratio
1	Performance	3	1
1	Usability	4	0.777777777777778
1	User Interfacce	3	1
1	Design	3	1
1	User Interface	3	1
2	Performance	4	1
2	Usability	4	1
2	User Interfacce	4	1
2	Design	4	1
2	User Interface	4	1

Figure 14: Enhanced Contingency

Figure-14 shows enhanced contingency which has the bug ids namely Bug1 and Bug2, Positive Category Ratio and Other Category Ratio are also computed for each category name.

L. Classifier Information



Classifier Information

Bug ID	Cat Name
1	Usability
2	Performance
2	Usability
2	User Interfacce
2	Design
2	User Interface

Figure 15: Classifier Information

Figure-15 shows the classified information. As shown in the Fig-15 each bug belongs to either single category or multiple categories. Like this the output for huge number of bugs.

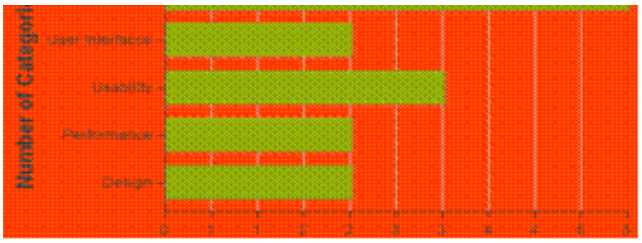


Figure 16: Bug classification

Figure-16 shows the classification of bugs under various categories. As shown in the Fig-16 based on the classification algorithm 2 bugs belong to design, 2 bugs belong to performance, 3 bugs belong to usability and 2 bugs belong to user interface.

M. Developer Registration

Figure-17 shows the registration process used by the developer. The developer provides various fields namely First name, Last Name, Desired User Name, Password, Email Id and the category which the developer mostly works on.

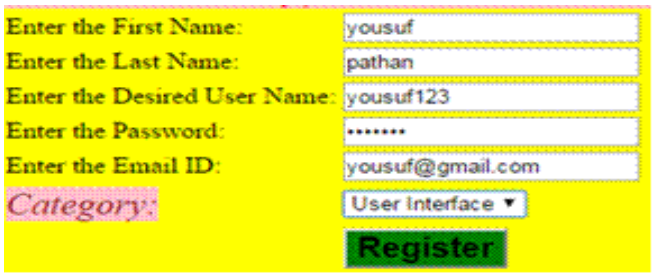


Figure 17: Registration

N. Developers Bug Assignment

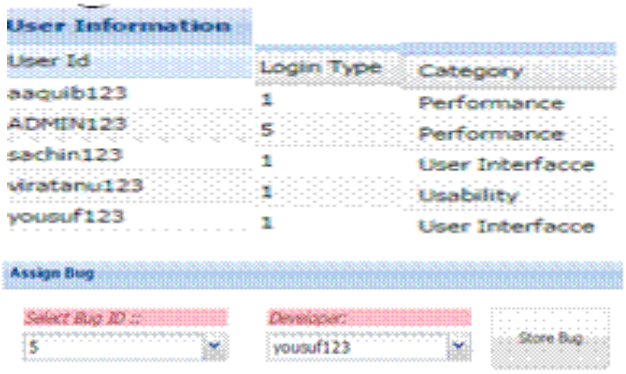


Figure 18: Bug Assignment

Figure-18 shows the Bug Assignment in which the bug id is being assigned to a developer. Here Bug Id is 5 and developer used is yusuf123.

O. Developers Bug

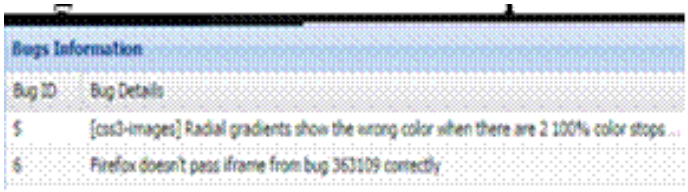


Figure 19: Developers Bug

Figure-19 shows the bug ids and details of the bugs assigned to the developers.

CONCLUSION

In this paper we have first taken bugs from standard products and applied series of algorithms like data cleaning, tokenization, frequency computation, Score computation and duplicate bug detection algorithm.

The bugs are also classified into various layers by computing the probability, contingency and enhanced contingency and finally applying the classifier.

FUTURE SCOPE

1. The work can be extended to support more products.
2. The classification can be also done graphically using k means algorithm along with applying the algorithm described in the paper for more accuracy.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?” in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.

[2] S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, “Finding bugs in web

applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36,no. 4, pp. 474–494, Jul./Aug. 2010.

[3] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," ACM Trans. Soft. Eng. Methodol., vol. 20, no. 3, Bugs 10, Aug. 2011.

[4] C.C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.

[5] Bugzilla, (2014). [Online]. Avaialble: http://bugzilla.org/

[6] K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, Aug. 2006, pp. 43–50.

[7] P.S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," IEEE Trans. Knowl. Data Eng., vol. 24, no. 6, pp. 1146–1150, Jun. 2012.

[8] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.

[9] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in Proc. ACM Conf. Comput. Supported Cooperative Work, Feb. 2010, pp. 301–310.

[10] V. Bol_on-Canedo, N. S_anchez-Maro~no, and A. Alonso-Betanzos, "A Bug of feature selection methods on synthetic data," Knowl.Inform. Syst., vol. 34, no. 3, pp. 483–519, 2013.

[11] V. Cerver_on and F. J. Ferri, "Another move toward the minimumconsistent subset: A tabu search approach to the condensed nearest neighbor rule," IEEE Trans. Syst., Man, Cybern., Part B, Cybern.,vol. 31, no. 3, pp. 408–413, Jun. 2001.

[12] D.Cubrani_c and G. C. Murphy, "Automatic bug triage using text categorization," in Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng., Jun. 2004, pp. 92–97.

[13] Eclipse. (2014). [Online]. Available: http://eclipse.org/

[14] B. Fitzgerald, "The transformation of open source software," MIS Quart., vol. 30, no. 3, pp. 587–598, Sep. 2006

[15] G. Lang, Q. Li, and L. Guo, "Discernibility matrix simplification with new attribute dependency functions for incomplete information systems," Knowl. Inform. Syst., vol. 37, no. 3, pp. 611–638, 2013.

[16] D. Lo, J. Li, L. Wong, and S. C. Khoo, "Mining iterative generators and representative rules for software specification discovery," IEEE Trans. Knowl. Data Eng., vol. 23, no. 2, pp. 282–296, Feb. 2011.

[17] Mozilla. (2014). [Online]. Available: http://mozilla.org/

[18] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in Proc. 6th Int. Working Conf. Mining Softw. Repositories, May 2009, pp. 131–140.

[19] G. Miao, L. E. Moser, X. Yan, S. Tao, Y. Chen, and N. Anerousis, "Generative models for ticket resolution in expert networks," in Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2010, pp. 733–742.

[20] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in Proc. 34th Int. Conf. Softw. Eng., 2012, pp. 25–35.

[21] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," Comput., vol. 42, no. 8, pp. 55–62, Aug.2009.

[22] Y. Yang, "An evaluation of statistical approaches to text categorization," Inform. Retrieval, vol. 1, pp. 69–90, 1999.

[23] Y. Yang and J. Pedersen, "A comparative study on feature selection in text categorization," in Proc. Int. Conf. Mach. Learn., 1997, pp. 412–420.

[24] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: An empirical study of commercial software projects," in Proc. 35th Int. Conf. Softw. Eng.,May 2013, pp. 1042–1051.

24] W. Zou, Y. Hu, J. Xuan, and H. Jiang, "Towards training set reduction for bug triage," in Proc. 35th Annu. IEEE Int. Comput. Soft. Appl. Conf., Jul. 2011, pp. 576–581.

[25] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy,"Characterizing and predicting which bugs get reopened," in Proc. 34th Int. Conf. Softw. Eng., Jun. 2012, pp. 1074–1083.

[26] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schr€oter, and C. Weiss, "What makes a good bug report?" IEEE Trans. Softw. Eng., vol. 36, no. 5, pp. 618–643, Oct. 2010.

[27] H. Zhang and G. Sun, "Optimal reference subset selection for nearest neighbor classification by tabu search," Pattern Recognit., vol. 35 pp. 1481–1490, 2002.

[28] X. Zhu and X. Wu, "Cost-constrained data acquisition for intelligent data preparation," IEEE Trans. Knowl. Data Eng., vol. 17, no. 11, pp. 1542–1556, Nov. 2005.

[29] T. H. Cheng and C. P. Wei, "A clustering-based approach for integrating document-category hierarchies," IEEE Trans. Syst., Man, Cybern. A,Syst., Humans, vol. 38, no. 2, pp. 410–424, Mar. 2008.

[30] H. C. Yang and C. H. Lee, "A text mining approach for automatic construction of hypertexts," Expert Syst. Appl., vol. 29, no. 4, pp. 723–734, Nov. 2005.

[31] M. Nagy and M. Vargas-Vera, "Multiagent ontology mapping framework for the semantic web," IEEE Trans. Syst., Man, Cybern. A, Syst., Humans, vol. 41, no. 4, pp. 693–704, Jul. 2011.

[32] W. Fan, D. M. Gordon, and P. Pathak, "An integrated two-stage model for intelligent information routing," Decis. Support Syst., vol. 42, no. 1, pp. 362–374, Oct. 2006.

[33] G. H. Lim, I. H. Suh, and H. Suh, "Ontology-based unified robot knowledge for service robots in indoor environments," IEEE Trans. Syst., Man Cybern. A, Syst., Humans, vol. 41, no. 3, pp. 492–509, May 2011.

[34] C. Lu, X. Hu, and J. R. Park, "Exploiting the social tagging network for web clustering," IEEE Trans. Syst., Man, Cybern. A, Syst., Humans, vol. 41, no. 5, pp. 840–852, Sep. 2011.

[35] A. J. C. Trappey, C. V. Trappey, F. C. Hsu, and D. W. Hsiao, "A fuzzy ontological knowledge document clustering methodology," IEEE Trans.Syst., Man, Cybern. B, Cybern., vol. 39, no. 3, pp. 806–814, Jun. 2009.

Author's Biography:

Dr. Kavitha K S is currently as an Professor in the department of Computer Science and Engineering at Global Academy of Technology, Bangalore, India. She received her Ph.D in Data Mining from the University of Anna in 2015,chennai, India.

Miss Rajeshwari M R is currently as an Assistant Professor in the department of Computer Science and Engineering at Ghousia College of Engineering, Ramanagara, Bangalore, India. She received her M.Tech in Computer Science and Engineering from the University of VTU in 2007,belguam, India.