iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** VI **Month of publication:** June 2018

DOI: http://doi.org/10.22214/ijraset.2018.6019

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue VI, June 2018- Available at www.ijraset.com

An Automated Approach for Mapping Bug Reports to Source Code and Bug Triaging

Alphy Jose[1], Aby Abahai T[2]
[1]*Computer Science and Engineering, Mar Athanasius College of Engineering and Technology, Ernakulam, Kerala, India*
[2]*Computer Science and Engineering, Mar Athanasius College of Engineering and Technology, Ernakulam, Kerala, India*

Abstract: ***The bug means the coding mistake that occurs in the software developing stage. It may occurs because of many reasons and some of the reasons are version mismatch, network incompatibility, and unavailability of supporting documents. Then the system will cause incorrect or unexpected behavior in the program. And bug report means a user level description about a bug, which is affected on the computer program. A bug report mainly having a bug id, summery about a bug and a detailed description about the bug. A tool for ranking all the source files with respect to how likely they are to contain the cause of the bug would enable developers to narrow down their search and improve productivity. The ranking is done on the basis of comparing the source code and the bug report, here 19 features are considering for the bug mapping procedure. And bug triaging refers to the process of assigning a bug to the most appropriate developer in order to fix the bug. The process of bug triaging is based on the interest of the developer and the bug mapping history of each developer. And also avoiding the chances of occurrence of duplication in repository. This method is very useful for java projects working in the Netbeans, Eclipse, Tomcat platforms.***

Keywords: ***Bug Report, Bug mapping, Bug triaging.***

I. INTRODUCTION

Software bug which results in an incorrect output or unexpected output due to the error or failure in a computer program. To permanently cure a bug we need to change the program. New bugs can be introduced due to the bug fixing process, so it should be the one of the most important step. Most of the cause of the bug are due to the mistakes, errors or due to the components in the operating systems, unavailability of the supporting documents, network incompatibility. Some of them are due to the incorrect code, which is produced by the compiler. Buggy means a program will be containing a huge number of bugs and the will be adversely affecting the functionality of the program. Under a testing environment while in the testing phase when testing the software which is found out by the testers are list of bugs are known as bug report or issue report. The test environment will be similar to the original environment. In the development site the test environment is created similar to the actual environment in which the software is supposed to work or run in live scenario. Bug reports which is used for understanding the developers about the software product defects. Majority of the companies spend their time in resolving the bugs during their day-to-day process. The software companies will be having different teams and these teams will be receiving a large number of bugs. One of the most difficult tasks is that the finding the location of source files with the correct bug. In their daily process as they are receiving a large number of bug reports and it is challenging for them to analyse manually debug and resolve them. So here introducing an automatic system that can rank the source code files with the relevant bug reports. From the source code will be taking the summary and description. Code and comments are extracted from the source code. This paper, which describes a methodology learning to rank files that is, ranking score, is computed by the weighted combination of the features. Features which specifies the relationship between the source code and bug report. Weights are trained on previously fixed bug reports. Here finding the similarity between the bug reports and the source code files and its methods, API similarities between the bug reports and source code files, semantic similarity between the bug report and source code files, computing collaborative score for recommending systems, bug fixing history, code change history, page rank score, hubs and authority score and local graph features by the dependency graph. That is obtaining ranking as which the pages that can occur the bug is being retrieved effectively. And also method for removing the duplicate bug reports. Manual bug assigning to the correct developer is expensive and usually results in wrong assignment of bug reports to developers. Proposing a method to automatically assign the bug reports to the correct developers by data reduction technique by feature selection that is, improving the quality of bug data. From the historical data sets we will be retrieving the attributes and constructing model that predicts the new bug set. We first applies feature selection technique to preprocess the textual information in bug reports, and then applies text mining technique build statistical models. The approach also includes the usage of the clustering to group the similar bug reports instead of random grouping that make it easy to assign the bug to the appropriate developer. For this process to take

place, we have to label the clustered groups in the order of prioritization. Then, the labeled groups will be assigned to the correct developer based on the domain knowledge. The purpose of doing this automation is that if we are considering an example eclipse which will be created by a group of developers. When a bug is occurred that is it will be a bug which is not fixed. To assign whom is a huge work. This process is having overhead. Developers will be working on different modules. So to identify a particular person we should take the previous history, current and we should communicate with peer developers and users. After that we should recreate the problem from that only we can identify the bug. This is time consuming to assign the bug to correct developer within a short span of time. And also expenditure will be also high. Thus we are developing an effective bug system that is finding the relevant pages that can occur the bugs, removing the duplicate bugs, and assigning this ranked pages to the correct developers so they can fix the bug fastly and accurately which can reduce the time consuming. The bug triaging will done based on the pattern matching algorithms, A fundamental assumption for these approaches is that the documents in the collection are all about one topic. However, in reality users' interests can be diverse and the documents in the collection often involve multiple topics. Topic modelling, such as Latent Dirichlet Allocation (LDA), was proposed to generate statistical models to represent multiple topics in a collection of documents, and this has been widely utilized in the fields of machine learning and information retrieval, etc. Here perform experiments on six large scale open source java projects namely, Eclipse, Aspectj, Tomcat, SWT, JDT, Birt.

II. LITERATURE SURVEY

The paper 'Improving bug localization using structured information retrieval' which is written by Saha[1]. Here uses Bluir method in which source code will be taken as the input and then we will be creating abstract syntax tree (AST) using JDT (Java development toolkit) and parsing through the abstract syntax tree. Dividing the source code into four document fields class, variable, comment, and method.

Then performing tokenization splitting into a bag of words using white spaces. And will be stored in the structured Xml document. Then it will be Units indexed into an array using an indexer. From the bug report extracting the summary and description. Performing tokenization as discussed above which is splitting into tokens by a bag of words using the white spaces.

Bluir which outperforms bug locator and here computing the similarity between the features as a single sum is having less accuracy than our method. In this method using the fixed revision of source code is used for the evaluation of bug reports which can lead to very bad contamination bug reports in case of future fixing bug information.

Next paper 'Where Should the Bugs Be Fixed?' which is written by Zhou[2]. Here propose a bug locator which is a method for retrieving the information. This is done for finding the location of the bug files. This method ranks all files that is having a textual similarity between the bug report and the source code file using the vector space representation model(VSM). When bug is received we will be computing the similarity between the bug and source code using the similarity measures by analyzing the past fixed bugs. The ranked list of files will be in decreasing order.

The top in the list are more likely to contain the result. If contains similar bugs then they are proposing another method that is, three layer heterogeneous graph.

First layer which represents the bug reports. Second layer shows previously reported bug reports, and the last layer which is the third layer which represents the source code files. Major disadvantages to the work are if the developer uses non-meaningful names the performance will be severely gets affected. And also bad reports which can cause misleading of the information and also essential information can cause significant delay.

And thereby performance will be affected. Next paper 'Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation' written by Xin ye[3] in this it is being done by using learning to rank algorithm. The ranking score is computed similarity between the source code files and the bug report. So for that using the feature extraction, extracting 19 features.

III. PROPOSED METHOD

For mapping bug reports to source code, first some preprocessing will done on the source code and the bug report. The source code contain the the code for the program and the commented description abiut a program. So we wants to perform the preprocessing on the code and the comments. In case of bug report it contain the summery about a bug and the description about a bug. The preprocessing on the bug report means, do all the preprocessing steps on the summery and the description. The below showing a bug report it having a bug id, summery and the description about the bug.

Bug ID: 339286
Summary: Toolbars missing icons and show wrong menus.
Description: The toolbars for my stacked views were: missing icons, showing the wrong drop-down menus (from others in the stack), showing multiple drop-down menus, missing the min/max buttons ...

Figure 1: Sample bug report

A. Preprocessing

Preprocessing in which knowledge extraction is being done. From the bug report use both description and summary. From the source code file use the whole content code and comments. For tokenization we will be splitting into words by using the white spaces. Then we remove thee stop words, punctuation, numbers etc. all words are reduced using porter stemmer as the NLTK[1] package. And by using vector space modeling find out the vector values of each term in a document. By developing a vocabulary of the terms in a document.

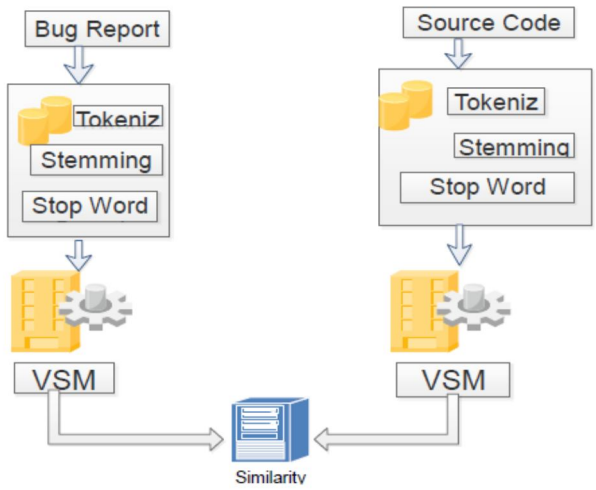
In the preprocessing stage first step is to tokenize the bug report and source code then removing the white spaces and special characters in the code and the report. Then by using the If we regard the bug report as a query and the source code file as a text document, then we can employ the classic vector space model (VSM) for ranking, a standard model used in information retrieval. In this model, both the query and the document are represented as vectors of term weights. Given an arbitrary document d (a bug report or a source code file), compute the term weights for each term t in the vocabulary based on the classical tf.idf weighting scheme in which the term frequency factors are normalized. The term frequency can be determined by finding the number of occurrence of a term in a document based on the total number of terms in a document.

B. Surface Lexical similarity

$$sim(r,s) = cos(\mathbf{r},\mathbf{s}) = \frac{\mathbf{r}^T\mathbf{s}}{\|\mathbf{r}\|\|\mathbf{s}\|}.$$

In the above equation we finding the surface lexical similarity between the terms in the bug report and the source code. For that we using the cosine similarity function. And ‘r’ in the above equation denoting the bug report and the ‘s’ denotes the source code.

For a bug report, we use both its summary and description to create the VSM representation. For a source file, we use its whole content—code and comments. To tokenize an input document, we first split the text into a bag of words using white spaces. We then remove punctuation, numbers, and standard IR stop words such as conjunctions or determiners. Cosine similarity function is used for checking the similarity checking between the source code and the bug report.



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue VI, June 2018- Available at www.ijraset.com

C. API enriched lexical similarity

Here find out the sematic similarity between the source code and the bug report is done. Which means some library function which including the information about button and user interfacing tools so such errors in the functions can be identified by using this Api enriched lexical similarity.

D. Collaborative Filtering Score

The file has be fixed before certain type of errors it can be identified by using this method consequently it is expected to be beneficial in our retrieval setting, too.

E. Class name similarity

Finding the class name similarity between the source code and the bug report. This feature having the high weightage than the all other feature evaluation technique. Both the summary and Description is used for the similarity checking.

F. Other features

1) *Bug –Fixing Recency:* It will deals about the recent bug that are fixed by the system. Which means the system will check whether it will fix the same bug. Such a way can improve the execution speed of the overall system
2) *Bug-Fixing Frequent :* Here check the frequent bug that fixed before
3) *Summery class name Similarity :* Checking the class name similarity between the summary of the bug report and the source code
4) *Summary method name similarity:* Here checking the similarity between the method name similarity in the summary of the bug report and the corresponding source code.
5) *Summary variable name similarity :* Here checking the similarity between the variable name similarity in the summary of the bug report and the corresponding source code
6) *Summary Comment name similarity :* Here checking the similarity between the comment name similarity in the summary of the bug report and the corresponding source code
7) *Description class name similarity:* Here checking the similarity between the class name similarity in the description of the bug report and the corresponding source code
8) *Description method name similarity :* Here checking the similarity between the method name similarity in the description of the bug report and the corresponding source code
9) *Description variable name similarity :* Here checking the similarity between the variable name similarity in the description of the bug report and the corresponding source code
10) *Description Comment name similarity :* Here checking the similarity between the comment name similarity in the description of the bug report and the correspondi.
11) *In-Link Dependencies :* In-Link frequencies are defined the page rank score, If the number of links connected to the class increases then the complexity of the code will also get increased.

in the information that they held, but were used as compilations of a broad catalog of information that led users direct to other authoritative pages.

In other words, a good hub represented a page that pointed to many other pages, and a good authority represented a page that was linked by many different hubs.

Page rank score determine the complexity of a source code and it is based on the in-link and out-link dependencies.

The hub score and the authority score are based on the Hyper Linked Induced Algorithm.

G. Weight Computation

For this we are using TF-IDF for calculation. TF which indicates the number of occurrences of specific term in the document.

IDF which indicates the number of documents that contain the specific term.

After the TF-IDF calculation cosine similarity is being done. Cosine similarity is the similarity between the bug report and the source code file.

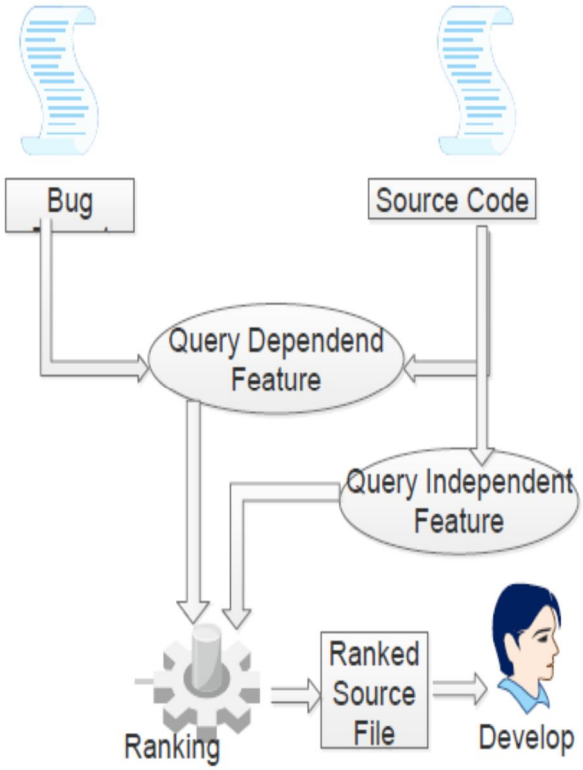


Figure 3: Bug mapping

$$f(r,s) = \mathbf{w}^T \mathbf{\Phi}(r,s) = \sum_{i=1}^{k} w_i * \phi_i(r,s).$$

In above introducing an equation the ranking model for f(r,s) is based on a weighted combination of features that capture domain dependent relationships between a bug report r and a source code file s The model parameters 'w' are trained using the learning to-rank approach from as implemented in the SVM. The aim of this formulation is to find a weight vector 'w' such that the corresponding scoring function ranks the files that are known to be relevant for a bug report at the top of the list for that bug report. **Φ**(r,s) indicating the feature measure that are evaluated using the all 19 features. And 'r' indicating the terms in the bug report and the 's' denoting the terms in the source code.

H. Semantic Similarity

Semantic similarity between two words which means that the two words whose meanings are similar. To find out the meaning between bug report and source code file we use machine learning approach. There are two phases: training phase and testing phase. The training which consist of bug reports and corresponding bug ids which indicates the semantic similarity between bug reports and source code files. Every bug reports in the training data which indicates the set of features. At training time, we range all bug reports and feature extraction functions to compile a feature vector per bug report. The feature vectors are stored in a matrix. We train a supervised learning method from the features and the bug ids of the training examples As the bug ids in the evaluation set that we use are binary, we build a classifier. At testing time, features are generated for the bug ids in the test set in a similar fashion as in the training phase, and a final prediction is made with the classifier trained in the training step.

I. Assigning Correct Developer

In this system we are developing a model to directly assign the bug report to the correct developer. The ranked list of pages that can occur as bug will be given to the correct developer. So for this process to occur we will be performing data reduction. That is reducing the data and also removing the duplicate bug reports. The architecture of the system is shown below.

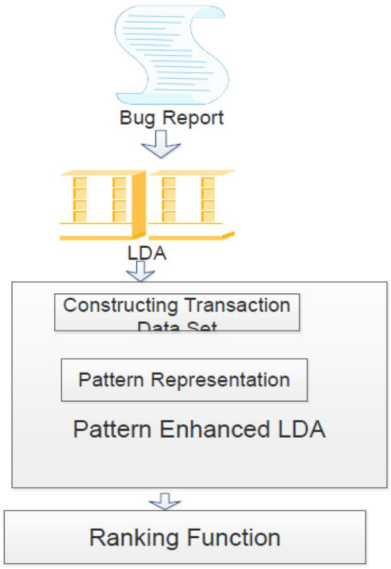


Figure 4: Assign to the Developer

IV. PATTERN BASED TOPIC MODEL FOR ASSIGN TO THE DEVELOPER

Many term-based or pattern-based approaches have been used in the field of information filtering to generate users' information needs from a collection of documents. A fundamental assumption for these approaches is that the documents in the collection are all about one topic. However, in reality users' interests can be diverse and the documents in the collection often involve multiple topics. In topic modelling, such as Latent Dirichlet Allocation (LDA), was proposed to generate statistical models to represent multiple topics in a collection of documents, which means that every document consisting of number of topics and these topics are used for the pattern matching algorithms, and this has been widely utilized in the fields of machine learning and information retrieval, etc. But its effectiveness in information filtering has not been so well explored. Patterns are always thought to be more discriminative than single terms for describing documents. However, the enormous amount of discovered patterns hinder them from being effectively and efficiently used in real applications, therefore, selection of the most discriminative and representative patterns from the huge amount of discovered patterns becomes crucial. To deal with the above mentioned limitations and problems, in this paper, a novel information filtering model, Maximum matched Pattern-based Topic Model (MPBTM), is proposed[11].

Topic		Z_1		Z_2		Z_3
Document	$\vartheta_{d,1}$	words	$\vartheta_{d,2}$	words	$\vartheta_{d,3}$	words
d_1	0.6	w_1, w_2, w_3, w_2, w_1	0.2	w_1, w_9, w_8	0.2	w_7, w_{10}, w_{10}
d_2	0.2	w_2, w_4, w_4	0.5	w_7, w_8, w_1, w_8, w_8	0.3	w_1, w_{11}, w_{12}
d_3	0.3	w_2, w_1, w_7, w_5	0.3	w_7, w_3, w_3, w_2	0.4	w_4, w_7, w_{10}, w_{11}
d_4	0.3	w_2, w_7, w_6	0.4	w_9, w_8, w_1	0.3	w_1, w_{11}, w_{10}

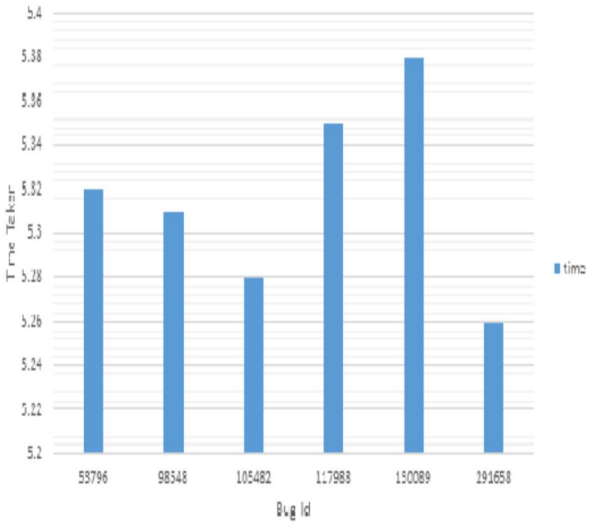
Fig 5: LDA topic Assignment

Pattern-based representations are considered more meaningful and more accurate to represent topics than word-based representations. Moreover, pattern-based representations contain structural information which can reveal the association between words. In order to discover semantically meaningful patterns to represent topics and documents, two steps are proposed: firstly, construct a new transactional dataset from the LDA model results of the document collection D; secondly, generate pattern-based representations from the transactional dataset to represent user needs of the collection D.

V. CONCLUSION

Through this work introduced an automated bug system which can be effectively used in the software companies. We will be getting ranked list of pages that can occur the bug and it will be automatically assigned to the correct developer who has developed the code. And also remove the duplication of the bugs. And also computed the semantic similarity between the bug report and source code file. From the previous experiments it was proved that learning to rank approach is having higher accuracy which is being used in our system. In the future work we can use additional types of domain knowledge such as stack traces and also features used in the defect prediction system. Also plan to use ranking svm in nonlinear kernels. Also to find how to prepare high quality datasets.

VI. PERFORMANCE EVALUATION



Here the performance of the system is evaluated. And the systems execution time needed for different bugs are calculated. The time needed for the execution is different
for different bugs. The below figure showing performance evaluation of the some bugs. In a classification task, precision for a class is the number of true positive divided by the total number of elements labeled as belonging to to positive class . Recall is in this context is defined as the number of true positive is divided by the total number of elements that are actually belonging to the positive class. In information retrieval a perfect precision score of 1.0 means that every result retrieved by a search was relevant whereas a perfect recall score of 1.0 means that all relevant documents where retrieved by a search, nothing about how many irrelevant document where also retrieved. The performance evaluation for the system was evaluated. As compared with the previous method, this method which having high accuracy than all others.
The Performance of the system will vary according to the size of the code. If the size of code is very high then the time taken for finding the buggy code is going to be high. And the bug triaging time will remains the same. And here showing the x and y axis of the graph plot based on the time taken for the execution and the corresponding bug report, y axis is high at the large sized bug report.

REFERENCES

[1] R. Saha, M. Lease, S. Khurshid, and D. Perry, “Improving bug localization using structured information retrieval,” in Proc. IEEE/ACM 28th Int. Conf. Autom. Softw. Eng., Nov. 2013, pp. 345–355. [2]

[2] J. Zhou, H. Zhang, and D. Lo, “Where should the bugs be fixed? –more accurate information retrieval-based bug localization based on bug reports,” in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2012 pp. 14–24. Xin Ye, “Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine Grained Benchmark, and Feature Evaluation” IEEE Trans. Softw. Eng., Vol. 42, No. 4, pp. 379-402, April. 2016.

[3] http://www.nltk.org/api/nltk.stem.html.

[4] G. Antoniol and Y.-G. Gueheneuc, “Feature identification: A novel approach and a case study,” in Proc. 21st IEEE Int. Conf. Softw. Maintenance,Washington, DC, USA, 2005, pp. 357– 366.

[5] G. Antoniol and Y.-G. Gueheneuc, “Feature identification: An epidemiological metaphor,” IEEE Trans. Softw. Eng., vol. 32, no. 9, pp. 627–641, Sep. 2006. B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala,

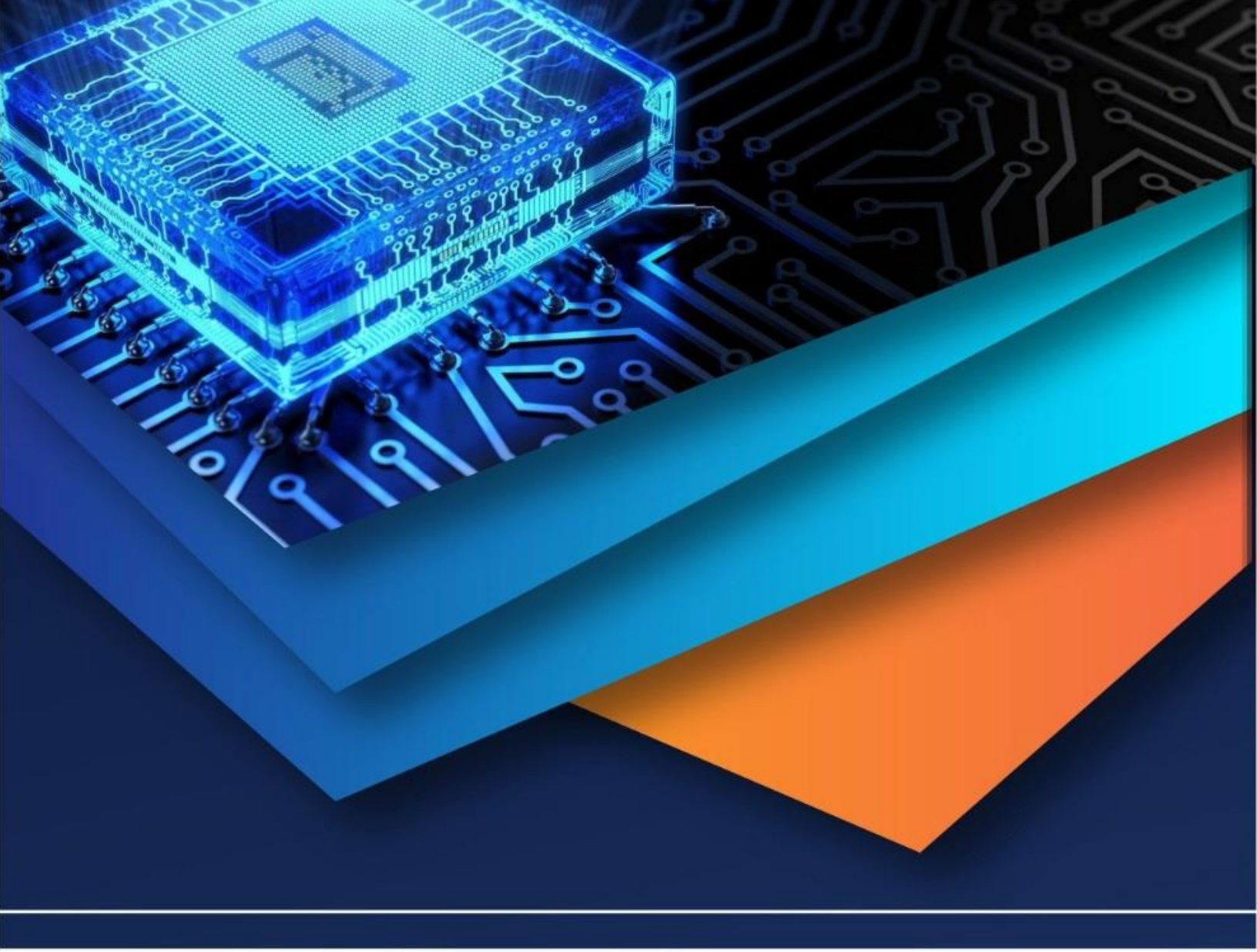
[6] "Debugadvisor: A recommender system for debugging," in Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng., New York, NY, USA, 2009, pp. 373–382.

[7] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in Proc. Int. Conf. Softw. Eng., Piscataway, NJ, USA, 2013, pp. 712–721.

[8] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in Proc. 18th ACM SIGSOFT Int. Symp. Found. Softw. Eng., New York, NY, USA, 2010 pp. 157–166.

[9] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Looking for bugs in all the right places," in Proc. Int. Symp. Softw. Testing Anal., New York, NY, USA, 2006, pp. 61–72

[10] Yang Gao,Yue Xu and Yuefeng Li "Pattern based topics for document modeling in information filtering" in proc.IEEE Transactions on knowledge and data engineering in 2013.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY