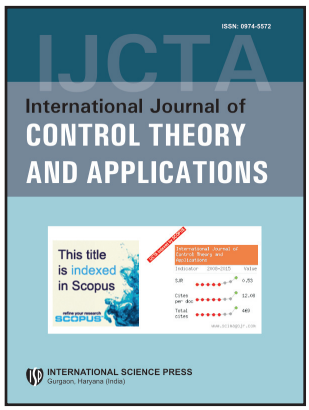

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 9 • Number 43 • 2016

An Efficient Approach for Automated Bug System

Sheryl Maria Sebastian[a], Asst Prof Neethu Subash[b] and Asst Prof Ani Sunny[c]

[a-c]Department of Computer Science and Engineering, M.A College of Engineering, Kothamangalam, Kerala, India. Email: [a]sherylmariasebastian@gmail.com; [b]neethu.subash@gmail.com; [c]anisunny88@gmail.com

Abstract: Software developing companies spends a lot of time consumed in tracking of bugs in their software. Locating the source code files with the corresponding bugs is a difficult task. It is challenging for developers to manually debug and fix them. Introduced an automatic system that can rank the source code files with bug reports. Describes a methodology learning to rank files that is, ranking score is computed by the weighted combination of the features. 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, dependency graph. And also method for removing the duplicate bug reports. Manual bug assigning to the correct developer is expensive and 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 retrieving the attributes and constructing model that predicts the new bug set. We first applies feature selection technique (e.g. Chi-square) to preprocess the textual information in bug reports, and then applies text mining technique (e.g., naive Bayes) to build statistical models. 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. We perform experiments on six large scale open source java projects namely, Eclipse, Aspectj, Tomcat, SWT, JDT, Birt.

Keywords: Software engineering; Bug Reports; Learning to rank; Artificial Intelligence; Information retrieval.

1. 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. 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 this 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 (e.g. Chi-square) to preprocess the textual information in bug reports, and then applies text mining technique (e.g., naive Bayes) to 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. We perform experiments on six large scale open source java projects namely, Eclipse, Aspectj, Tomcat, SWT, JDT, Birt.

2. RELATED WORK

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

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 bugreports 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 buglocator 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 analysing 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 significiant 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.

3. PROPOSED WORK

First from the collection of bug reports we will be text preprocessing is being done. Then calculation of the weights calculation of the features and ranking of the bug reports are being done.

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 tokenisation 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.

B. 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.

C. 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 bugreports and corresponding bug ids which

indicates the semantic similarity between bugports and source code files. Every bug reports in the training data which indicates the a 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.

D. 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.

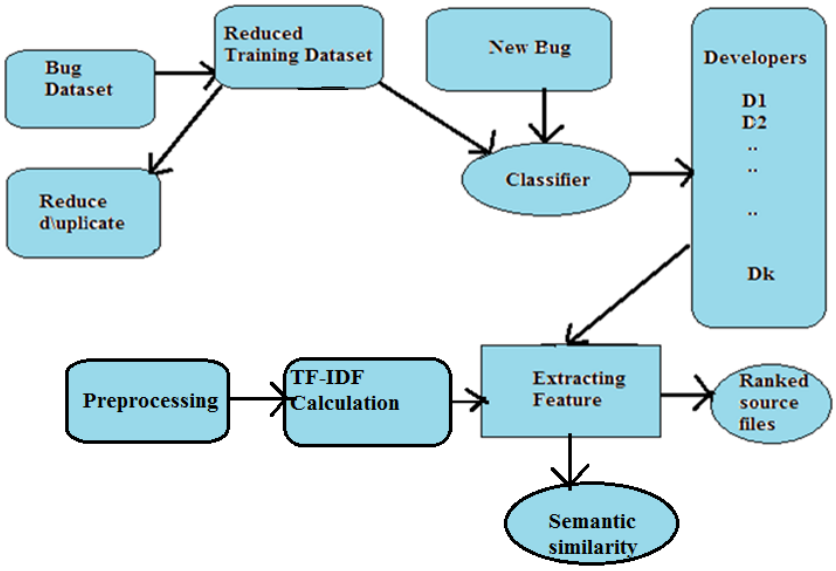

Figure 1: Architecture

E. Reduction of Data

This can be done by using feature selection techniques. Converting the bug data into a two- dimensional matrix. That is word and bug. Feature selection which is carried out by the chisqaure test which is defined as follows:

$$\Sigma \text{ et } \varepsilon(0,1),\ \text{ec}\varepsilon(0,1)\ \Sigma(\text{Net } \theta c - \text{Eet } \theta c)2/\text{Eet } \theta c$$

$$\text{et} = 1(\text{vocabulary contains term t})$$

$$\text{ec} = 1(\text{word is in class c})$$

$$N = \text{observed frequency of words}$$

$$E = \text{expected frequency of words.}$$

$$E = N * P(t) * P(c)$$

Expected frequency of *t* & *c* occurring:

$$E11 = N * (N11 + N10)/N * (N11 + N01)/N$$

Arithmetically,

$$X2(V, t, c) = (N11 + N10 + N01 + N00) * (N11N00 - N10\ N01)2$$
$$(N11 + N01)\ (N11 + N10)\ (N10 + N00)\ (N01 + N00)$$

The values are calculated. Selecting the top words and then generating the training dataset.

F. Text Categorization

Classification of documents (vocabulary) in to fixed number of predefined categories. Supervised machine learning algorithm is used to generate a classifier. eg: Naïve Bayesian Bug reports are instances, words are features & label (developer) indicates class.

G. Naive Bayes (nb)

Naive Bayes assumes that features (i.e., terms) are conditionally independent given a label. Based on Bayes Rule.

For a document d and a class c,

$$P(c|d) = P(d|c)\ .\ P(c)$$

P(d) Relies on Bag of Words representation of bug reports

To find most likely class we use

$$CMAP = \text{argmax}\ c\varepsilon CP(d|c).P(c)$$

Assuming the conditional independence between the term and class, we have

i.e., $$P(t1, \ldots, tn \mid c) = P(t1 \mid c) \bullet P(t2 \mid c) \bullet P(t3|c) \bullet \ldots \bullet P(tn \mid c)$$

$$CNB = \text{argmax}\ c\varepsilon CP(cj)\ .\ \Pi\ x\varepsilon X\ P(xj|c)$$

Learning the Multinomial Naïve Bayes Model

Prior Probability $P(c) = \frac{Nc}{N}$

Nc = no: of docs labelled as c, N = total no: of d

Conditional probability: fraction of times word appears among all words in document of classc, = P(tk|c) = Count(tk, c)/ΣtεV Count(w, c)

After Laplace Smoothing

$$CMAP = \text{argmax}\ c\varepsilon CP(t1, t2, \ldots)|c).P(c)$$

Best class can be found using

$$P(tk|c) = Count(tk, c) + 1/\Sigma t\varepsilon V\ Count(w, c) + |V|$$

Here extracting attributes for historical bug datasets and trained by using a naïve Bayesian classifier. Predicting the reduction order for a new bug dataset. Then applying the predicted reduction orders to the new bug dataset. Assigning the bug reports to the correct developers. The major advantage of naive Bayes classification is its short computational training time, since it assumes conditional independence. Notice that in naive Bayes, we only consider the presence or absence of a term in a bug report. The number of times a term appears in the bug report is not considered. From this classifier we can predict the bugs corresponding to the correct developer.

H. Removing Duplicate Reports

Removal of bug reports which will leads to more accuracy

Algorithm:

STEP 1: Start

STEP 2: Randomly selecting a bug report from dataset.

STEP 3: Similarity of bug report between each query and every document set is compared.

STEP 4: Bug reports having similarity greater than a threshold value is removed and others are kept inside the dataset (set threshold >0.8).

STEP 5: Do this until there exist no duplicate bug report in 2-3 iterations

STEP 6: Stop

And atlast use a svm classifier in which we give the input features to the classifier. Classifier which constructs a training model. Lucene which is used to index the features and from that when new bug report arrives it will predict the correct location.

I. Result

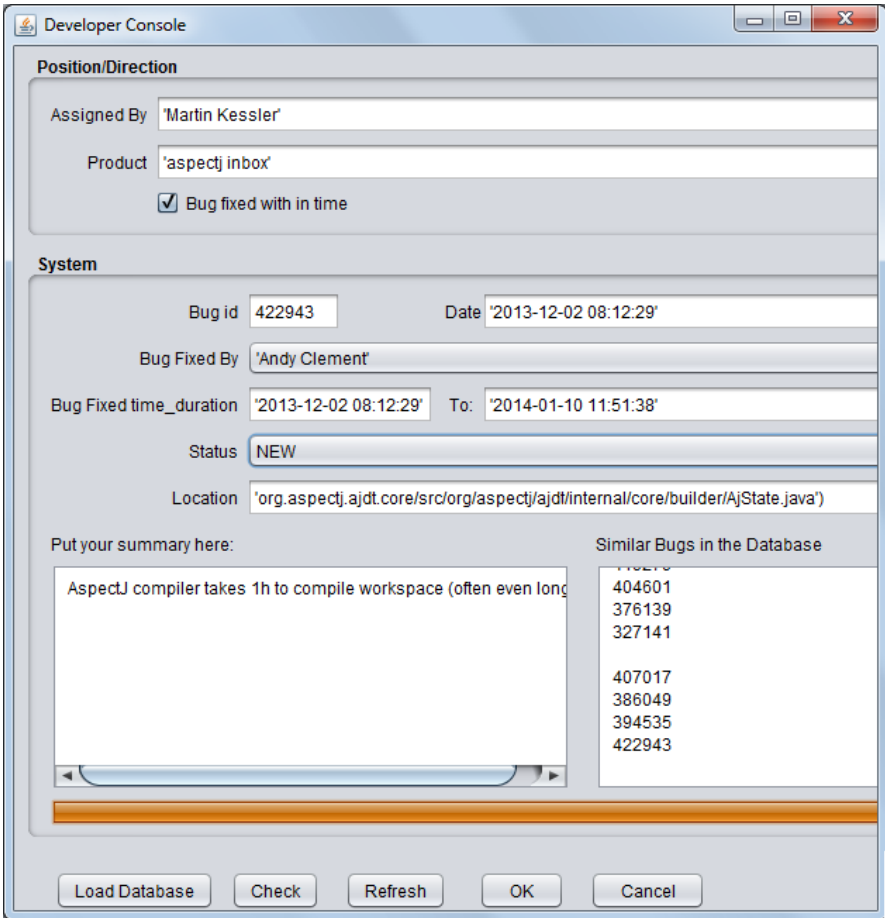

Figure 2: Result

```
(class:=!324932!324932!324932!324932
com/arjuna/ats/internal/jta/transaction/arjunacore/TransactionImple=!324932!3249
FILE:   C:\Users\Sheryl\Desktop\bugmapping\MappingBugReport\src\dataset\TreeStru
FILE:   C:\Users\Sheryl\Desktop\bugmapping\MappingBugReport\src\dataset\Pointcut
FILE:   C:\Users\Sheryl\Desktop\bugmapping\MappingBugReport\src\dataset\BrowserV
FILE:   C:\Users\Sheryl\Desktop\bugmapping\MappingBugReport\src\dataset\GoToLine
FILE:   C:\Users\Sheryl\Desktop\bugmapping\MappingBugReport\src\dataset\BuildPro
FILE:   org.aspectj.ajdt.core/src/org/aspectj/ajdt/internal/core/builder/AjState
tests/bugs175/pr423257/AspectX.java
tests/bugs175/pr423257/Test.java
tests/src/org/aspectj/systemtest/AllTests17.java
tests/src/org/aspectj/systemtest/ajc175/Ajc175Tests.java
tests/src/org/aspectj/systemtest/ajc175/AllTestsAspectJ175.java
tests/src/org/aspectj/systemtest/incremental/tools/MultiProjectIncrementalTests.
weaver/src/org/aspectj/weaver/bcel/asm/StackMapAdder.java
Reading model...done.
Reading test examples...done.
Classifying test examples...done
Runtime (without IO) in cpu-seconds: 0.00
Average loss on test set: 0.0000
Zero/one-error on test set: 0.00% (1 correct, 0 incorrect, 1 total)
NOTE: The loss reported above is the fraction of swapped pairs averaged over
      all rankings. The zero/one-error is fraction of perfectly correct
      rankings!
Total Num Swappedpairs  :      0
Avg Swappedpairs Percent:   0.00
Rank of files:2.36487995
Rank of files:1.83869983
Rank of files:1.67179438
Rank of files:1.86228258
Rank of files:1.39989559
```

Figure 3: Result with ranking of location

J. Benchmark Datasets

AspectJ13: an aspect-oriented programming extension for Java.

4. 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.

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] 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.

[3] 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.

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