ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

Using BM25 weighting and Cluster Shrinkage for Detecting Duplicate Bug Reports

Nhan Minh Phuc

Faculty of Information Technology, Tra Vinh University, Viet Nam

Abstract: In software maintenance, bug reports play an important role for the correctness of software packages. Unfortunately, a duplicate bug report problem arises because there are significant many duplicate bug reports in various software projects. Processing duplicate bug reports is thus time-consuming and has high cost of software maintenance. In this research, we propose a detection scheme based on the BM25 weighting and cluster shrinkage (BM25-CS) to enhance the detection performance. The effectiveness of this method is verified in an empirical study with three open-source projects, SVN, Argo UML, and Apache. The experimental results show that our method outperforms other detection schemes about 6-10% in all cases.

Keywords: Bug Reports, Duplication Detection, B25Weighting, Cluster Shrinkage

I. INTRODUCTION

In software maintenance, bug reports are an important resource to fix software bugs. For many software projects, such as Eclipse and Subversion, bug tracking systems (BTS) are used to receive bug reports in which reporters can describe, track, and classify observed software bugs. From these BTS repositories, a triager analyzes the incoming bug reports and then decides appropriate programmers to fix the bugs. As reported in [14], the triaging process is usually time-consuming. For example, around 30 minutes are estimated for processing a bug report on average in Sony Ericsson [14]. Automatic mechanisms are thus important for triagers to shorten the triaging time. In the triaging work, detecting duplicate bug reports is one of the important issues in recent research work [10, 14, 15, 16, 17, 20]. The main reason is that the number of the duplicate bug reports to the total bug reports is usually significant. For example, in [1] reported that 20% of the bug reports were duplicated in a collected Eclipse dataset of 18,165 bug reports. It similarity to Firefox dataset, 36% of total bug reports are duplicated. For Eclipse, Apache, and Fedora Core, the duplicate rates are 17%, 14%, and 13%, respectively. This means processing duplicate bug reports costs much precious triaging time.

In previous work, many automatic detection schemes have been proposed [7, 10, 11, 15]. They can be mainly classified into two categories: duplicate detection based on textual bug descriptions only [6, 10, 11, 14, 17, 20], and duplicate detection based on textual information and software execution information [18]. The schemes of the first category of using textual information mainly focus on exploring the implicit semantic meanings of the bug reports. For instance, Hiew proposes a clustering technique using TF- IDF (term frequency-inverse document frequency) to calculate the similarity for detection [10]. Runeson et al. propose a detection scheme using basic natural language processing (NLP) techniques, such as stemming and synonym expansion, to detect duplicate bug reports [14]. Sureka and Jalote then propose an n-gram scheme to improve the detection performance [16]. Therefore, the detection performance relies on the effectiveness of employed text mining, natural language processing, and information retrieval techniques. Due to the ambiguity of textual descriptions using natural languages, achieving high detection performance is a challenging research issue.

For the second category of detection schemes using both textual information and software execution information can be achieved high detection performance. For example, Wang et al. proposed a framework calculating different similarity metrics for textual information and execution information [18, 20]. Their experimental results show that this approach outperforms other approaches using textual information alone. However, collecting software execution information is not a straightforward task. Using execution information needs a large amount of additional storage space to store execution traces. In addition, generalizing this framework for different software projects may need many human efforts to customize the process for collecting information. Therefore, the feasibility is limited. In this paper, we focus on the improvement on the first category by proposing a BM25-CS scheme. In this scheme, the BM25 weighting information and cluster shrinkage are considered to improve the discriminative effectiveness of important textual features. It is an important part in support for duplicate detection scheme.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

Figure 1.1: An Example of a Duplicate Bug Report in SVN

II. PROBLEM DEFINITONS

In this research, duplicate bug reports are defined as the bug reports that describe the same bug issue as a master bug report. Figure 1.1 shows an example of a duplicate bug report #983 in software project Subversion (SVN). From the figure, we can find that bug report #983 is identified as a duplicate of its master bug report #88. Therefore, to solve the duplication detection problem for a given set of bug reports is to design a mechanism which can automatically identify duplicate bug reports.

Table 1.1: An example of master bug reports and their duplicate reports of SVN

Cluster No.	Master BR	Duplicate BR
1	689	793
2	703	792, 797
3	70	836
4	491	492, 897
5	88	983
...	...	...
N	872	968, 1048

In the previous studies, this problem is investigated by giving a duplication recommendation list for each incoming bug report [14]. Therefore, the historical bug reports are first classified into clusters according to their duplication relationship. Table 1.1 shows an example for SVN in which bug reports are classified into n clusters. Each duplicate bug report cluster has a master bug report. In this study, we use the earliest one without loss of generality. That is, the bug report of the smallest bug report ID is the master bug report. Then, for each incoming bug report, the recommendation list ranks the historical bug reports. The detection performance can then be decided with the recall rate which is the number of correct predictions to the number of total correct duplicates. The performance evaluation of duplication detection schemes can be conducted using their top-k recall rates. The research goal is to improve the top-k recall rate of the detection mechanism.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

III. DUPLICATE DETECTION SCHEME

Figure 3.1 illustrates the entire detection process in the proposed BM25-CS scheme. The detection flow has the following five steps.

A. Natural language processing (NLP) preprocessing: In this step, each term feature of the bug reports is extracted and processed with standard NLP preprocessing techniques, such as tokenization, stopword removal, and stemming. When an incoming bug report is submitted, both the historical bug reports and the incoming one are transformed into a vector space model after the standard NLP preprocessing steps, such as tokenization, stemming, and stopword removal **Error! Reference source not found.**. In NLP preprocessing, the Porter stemming algorithm **Error! Reference source not found.** is adopted for stemming and a standard stopword list is used with some special characters like ‘,’, ‘.’, ‘?’, ‘*’, and ‘$’ to remove common words.

Table 3.1: An example of mapping duplicate bug reports and the incoming

Cluster No.	Duplicate Bug Report	Incoming Bug Report
1	329	330
2	387	433
…	…	…
122	1382, 1482, 2287	2460
…	…	…

B. Cluster Shrinkage : We use the cluster shrinkage to help us find the semantics of bug report overlap. In this way, it will increase the member of cluster relationship by the threshold. The first, we have to find a center of cluster. The second, we shrink all of bug report to its center.

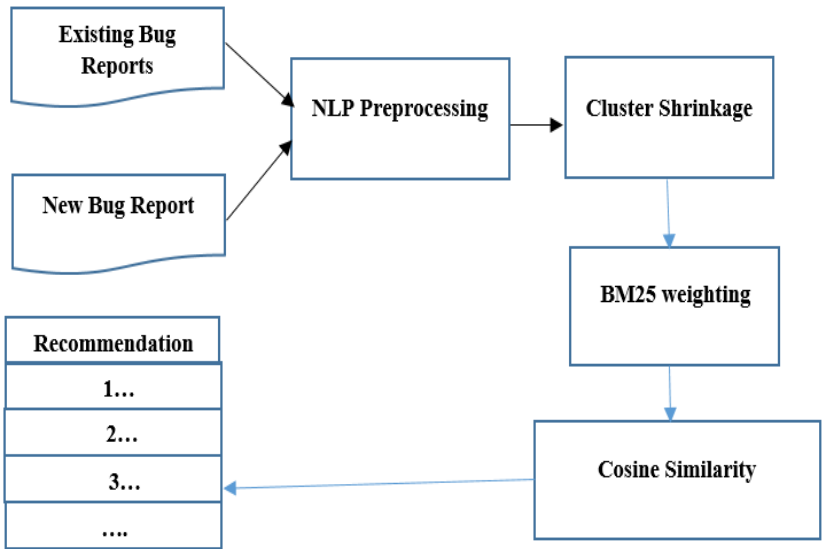


Figure 3.1: detection process in BM25-CS scheme

1. Centroid of Clusters: the centroid, we use it to represent the cluster, is a center vector. Each cluster has a centroid that with all information in its cluster. We use the average vector that in a cluster to calculate centroid. Because the submitter does not always describe the bug in detail, it will make the similarity calculation inefficient. The reason is two duplicate bug reports with seldom same words and it will make to determine whether they are duplicate bug report become difficult. So the centroid can help us increase similarity between duplicate bug reports, it have more words.

2. Using Cluster Shrinkage: After we find the each centroid of cluster, we shrink all of bug report to its centroid of cluster. The symbol is a threshold. The symbol v represents a bug report vector. The symbol v′ means the new vector.

For each cluster {
N is the number of bug reports in S

Compute its centroid:

$$C=\frac{1}{N}\sum^{N} v$$

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

For each bug report v ∈ S

{

v′=(1 - λ) v + λc

Where $0 \leq \lambda \leq 1$

}

Table 4.1: Datasets of three open-source projects

Description	ArgoUML	Apache	SVN
Language	Java	Java	C
Software Type	UML Tool	HTTP Server	SCM Tool
SCM	Subversion	Subversion	Subversion
Repository	Tigris	Bugzilla	Tigris
Data Period	00/02-07/05	01/01-07/02	01/03-07/05
# of Bug Reports	6.613	2.771	2.296
# of Duplicates	755	614	313

According to the duplication relationship information, the historical bug reports are classified into clusters (classes) for further detection operations. Table 3.1 shows an example of the clusters in project SVN. In Table 3.1, the bug report with the largest ID will be used as the testing bug report for evaluation. Figure 3.2 illustrates documents in each cluster are moved toward the cluster centroid c in cluster shrinkage.

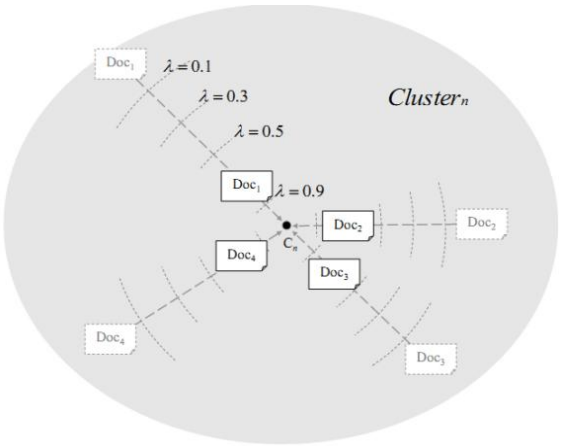


Figure 3.2: documents in each cluster are moved toward the cluster centroid c in cluster shrinkage

C. Enhanced BM25 weighting: In information retrieval, BM25 is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. In the case of processing duplicate bug reports, documents are ranked based on a bag-of-words retrieval function, and each term is treated as a query term to calculate term occurrences in all the documents. In addition, it can also be represented with slightly different components and parameters to adjust to respective information retrieval applications **Error! Reference source not found.**. The weighting function of BM25 for a query string q and a document d is defined as follows **Error! Reference source not found.**:

$$score\ (q,d) = \sum_{i=1}^{|q|} idf\,(q_i) \frac{tf\,(q_i,d)\cdot(k_1+1)}{tf\,(q_i,d)+k_1\cdot(1-b+b\cdot\frac{|d|}{dl_{avg}})} \quad (1)$$

Where $tf(q_i,d)$ is the term frequency q_i appears in the document d, $|d|$ is the length of d in words, and dl_{avg} is the average document length in the collection. The parameters k_1 and b are free parameters for controlling the weighting between the term frequency and the normalized document length. They are usually chosen as $1.2 \leq k_1 \leq 2.0$ and $0.5 \leq b \leq 0.8$. In this paper, we use the common settings of $k_1 = 2.0$ and $b = 0.8$ as other studies.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

Finally, idf(q_i) is the IDF (inverse document frequency) weight of the query term q_i. It is computed as:

$$idf(q_i) = \log\left(\frac{N - df(q_i + 0.5}{df(q_i) + 0.5}\right) \qquad (2)$$

where N is the total number of documents in the collection, and df(q_i) is the number of documents containing q_i.
In **Error! Reference source not found.**, BM25 can properly represent both the local weights and the global weights by considering the term frequency, the inverse document frequency, and the document length. For each term t_i in a bug report BR_j, its weight w_i is adjusted Accordingly as follows:

$$w_i = idf(t_i)\frac{tf(t_i, BR_j)\cdot(k_1+1)}{tf(t_i, BR_j) + k_1\cdot(1-b+b\cdot\frac{|BR_j|}{dl_{avg}})} \qquad (3)$$

where idf(t_i) is computed as:

$$idf(t_i) = \log\left(\frac{N - df(t_i) + 0.5}{df(t_i) + 0.5}\right) \qquad (4)$$

In (4) we adjust its value to a floor of 0 because it can return a negative value when the term t_i appears in more than half of the documents, this means that adjustment of the IDF function the common terms appearing in more than half of the bug reports are ignored. Only the terms appear in less than half of the documents are considered as the significant features to represent the document vectors.

D. Cosine similarity: Cosine measure is employed to compare the similarity between a new bug report and the cluster centroids. Due to Cosine measure in [2] has the best performance in comparison with the Dice coefficient, and the Jaccard index. The Cosine similarity is calculated by the following formula:

$$Cos(\vec{X}_i, \vec{Y}_j) = \frac{\vec{X}_i \cdot \vec{Y}_j}{|\vec{X}_i| \cdot |\vec{Y}_j|} \qquad (5)$$

where $\vec{X}_i$ represents the feature vector of an incoming bug report, and $\vec{Y}_j$ represents a centroid of existing bug reports in the dataset Y.

E. Top-n recommendation: We present the result like as previous work [4]. Using the top-n recommendation system can help user to find the duplicate bug reports. We list rank from 1 to 20 and observe the performance in every rank. Then, we compare the top-N recommendation with past researches. Finally, the recall rate is calculated as the detection performance in this study.

IV. EXPERIMENTAL RESULTS

A. Experimental Datasets: For the experimental datasets, we used three datasets of open source software projects, Apache, ArgoUML, and SVN. Table 4.1 lists the detailed information of these three datasets. To evaluate the BM25-CS scheme and related duplication detection schemes, the recall rate metrics in Equation (4.1) was used in the experiments. Equation (4.1) illustrates how to calculate the recall rate, where Ncorr is the number of duplicate reports that are correctly identified, and Ntotal is the total number of duplicate reports.

$$\text{Recall Rate} = \frac{N_{corr}}{N_{total}} \qquad (4.1)$$

It is defined as the percentage of the duplicates that can correctly find the corresponding master bug reports in the top-n recommendations.

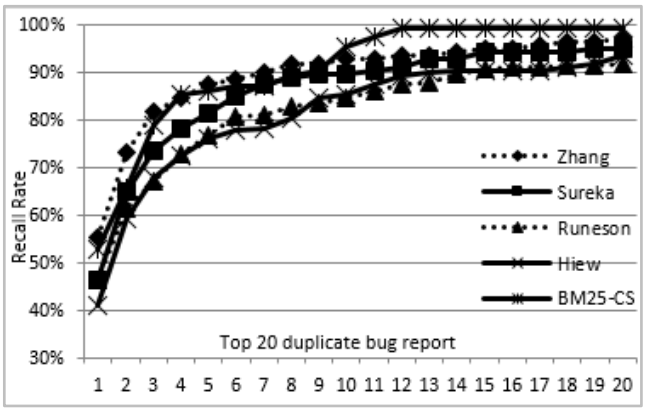
B. Comparison evaluation: To explore the effectiveness of BM25-CS in comparison with related detection schemes, experiments were conducted to study the work of Zang [19], the work of Sureka [16], the work of Runeson [14], Hiew [10], and BM25-CS. The results are shown in Figures 4.13. From the results, we can find that BM25-CS outperforms other detection schemes. The main reason is that the BM25 weighting combined with cluster chrinkage can effectively enhance the discrimibility for duplication detection. The results demonstrate its effectiveness.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

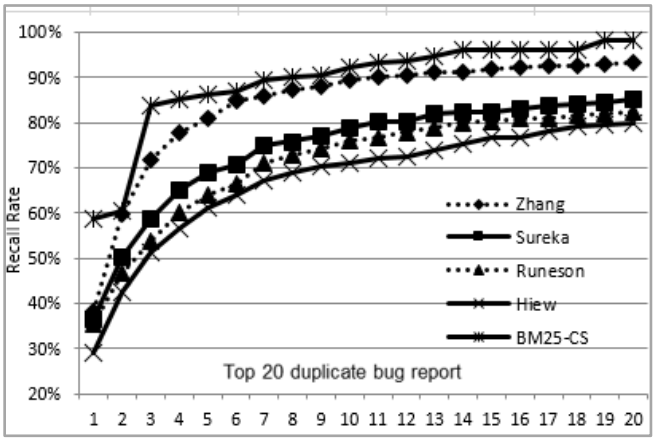
IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

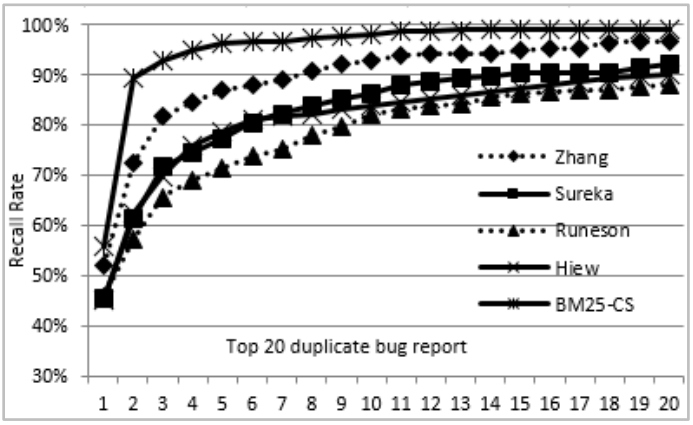
Vol. 7, Issue 11, November 2018



(a) SVN



(b) ArgoUML



(c) Apache

Fig. 4.13 Detection performance for three software projects: Apache, ArgoUML, and SVN

CONCLUSIONS

Duplication detection of bug reports is an important issue in software maintenance. This is because the same bugs or failures may be reported several times. Since the number of duplicate bug reports is significant in previous observations, identity it with a detection scheme can save much human effort and cost. In this research, we propose a detection scheme called BM25-CS using BM25 weighting to improve the detection performance. The experiments show that the BM25-CS scheme can achieve the best performance among several state-of the-art related schemes. There are still many issues to be discussed. For example, how to find the best parameter settings of BM25 weighting can be a challenging issue. However, our experiments show that some settings can achieve extremely high performance. This demonstrates the feasibility of BM25-CS.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 7, Issue 11, November 2018

REFERENCES

[1]. John Anvik, Lyndon Hiew, and Gail C. Murphy, "Coping with an Open Bug Repository," in Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange (eclipse '05), 2005, pp. 35–39.
[2]. Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann, "What Makes a Good Bug Report?" in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16), 2008, pp. 308–318.
[3]. Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim, "Duplicate Bug Reports Considered Harmful... Really?" in Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM 2008), 2008, pp. 337–345.
[4]. Yguaratã Cerqueira Cavalcanti, Paulo Anselmo da Mota Silveira Neto, Euardo Santana de Almeida, Daniel Lucrédio, Carlos Eduardo Albuquerque da Cunha, and Silvio Romero de Lemos Meira, "One Step More to Understand the Bug Report Duplication Problem", in Proceedings of the 24th Brazilian Symposium on Software Engineering (SBES'10), 2010, pp. 148–157.
[5]. Yguaratã Cerqueira Cavalcanti, Eduardo Santana de Almeida, Carlos Eduardo Al- buquerque da Cunha, Daniel Lucrédio, and Silvio Romero de Lemos Meira, "An Initial Study on the Bug Report Duplication Problem", in Proceedings of the 14th European Conference on Software Maintenance and Reengineering, 2010, pp. 264–267.
[6]. Zhi-Hao Chen, "Duplicate Detection on Bug Reports using N-Gram Features and Cluster Shrinkage", Master Thesis, Yuan Ze University, Jul. 2011.
[7]. Hung-Hsueh Du, "A Study of Duplication Detection Methods for Bug Reports based on BM25 Feature Weighting," Master Thesis, Yuan Ze University, Nov. 2011.
[8]. Hu Guan, Jingyu Zhou, and Minyi Guo, "A Class-Feature-Centroid Classifier for Text Categorization" in Proceedings of the 18th International Conference on World Wide Web (WWW 2009), 2009, pp. 201–210.
[9]. Eui-Hong Han and George Karypis, "Centroid-Based Document Classification: Analysis and Experimental Results," in Proceedings of the Fourth European Con- ference on Principles of Data Mining and Knowledge Discovery (PKDD '00), 2000, pp. 424–431.
[10]. Lyndon Hiew, "Assisted Detection of Duplicate Bug Reports," Master Thesis, The University of British Columbia, May 2006.
[11]. Nicholas Jalbert and Westley Weimer, "Automated Duplicate Detection for Bug Tracking Systems," in Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008), 2008, pp. 52–61.
[12]. Mostafa Keikha, Narjes Sharif Razavian, Farhad Oroumchian, and Hassan Seyed Razi, "Document Representation and Quality of Text: An Analysis," in Survey of Text Mining II, Michael W. Berry and Malu Castellanos, Eds.Springer London, 2008, ch. 12, pp. 219–232.
[13]. Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford, "Okapi at TREC-3," in Proceedings of the Third Text REtrieval Conference (TREC-3), 1994, pp. 109–126.
[14]. Per Runeson, Magnus Alexandersson, and Oskar Nyholm, "Detection of Duplicate Defect Reports using Natural Language Processing," in Proceedings of the 29th In- ternational Conference on Software Engineering (ICSE 2007), 2007, pp. 499–510.
[15]. Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo, "A Discriminative Model Approach for Accurate Duplicate. Bug Report Retrieval," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engi- neering (ICSE 2010), vol. 1, 2010, pp. 45–54.
[16]. Ashish Sureka and Pankaj Jalote, "Detecting Duplicate Bug Report using Character N -Gram-based Features," in Proceedings of the 17th Asia Pacific Software Engi- neering Conference (APSEC 2010), 2010, pp. 366–374.
[17]. Minh, P.N.: An approach to detecting duplicate bug reports using n-gram features and cluster chrinkage technique. Int. J. Sci. Res. Publ. (IJSRP) 4(5), 89–100 (2014).
[18]. Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun, "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Informa- tion," in Proceedings of the 30th International Conference on Software Engineering (ICSE '08), 2008, pp. 461–470.
[19]. Xiaoyan Zhang, Ting Wang, Xiaobo Liang, Feng Ao, and Yan Li, "A Class-based Feature Weighting Method for Text Classification," Journal of Computational Infor- mation System, vol. 3, pp. 965–972, 2012.
[20]. Phuc NM. Improving detection performance of duplicate bug reports using extended centroid features. International Journal of Advanced Research in Computer and Communication Engineering. 2014 Oct; 3(10):8252–7

BIOGRAPHY

P Nhan Minh - received the B.Sc in Information Technology from the Ho Chi Minh City University of Natural Sciences-Vietnam National University, and M.Sc in Computer Science and Engineering from Yuan Ze University, Taiwan. He is now working in Tra Vinh University, Viet Nam.