2011 11th IEEE International Conference on Computer and Information Technology

A Bug Rule based Technique with Feedback for Classifying Bug Reports

Tao Zhang
School of Computer Science
The University of Seoul
Seoul, Korea
kerryking@ieee.org

Byungjeong Lee*
School of Computer Science
The University of Seoul
Seoul, Korea
bjlee@uos.ac.kr

Abstract—**As software programs become increasingly large and complex, it is more important to improve the quality of software maintenance. Many software programs rely on bug reports to correct errors in maintenance activities. Bug tracking systems were developed to guide maintenance activities of software developers. However, due to the excessive number of duplicate bug reports, developers spend much time to identify these bug reports. In this study, in order to save developers' time in software maintenance, we propose a bug rule based classification technique to categorize bug reports. By utilizing developer feedback mechanism in the technique, it distinguishes duplicate and valid bug reports and is expected to improve the accuracy of bug reports retrieval. Finally, we show the feasibility of this technique in experiment and case study.**

Keywords-bug report classification; bug tracking system; developer feedback; software maintenance.

I. INTRODUCTION

With huge amounts of software projects, it becomes more difficult to check bugs from a large number of source code files. Essential activities in software maintenance include bug reporting and fixing [1]. Maintenance activities of many software projects rely on bug reports to correct defects in source code files [2]. Recently, plenty of software companies use bug tracking systems [3] during software development to track bugs, where developers can correct these bugs quickly. Bug tracking systems allow developers to upload, describe and comment on bug reports. Bugzilla [4][5], a web-based bug tracking tool, is a famous open source bug tracking system. Developers can submit bug reports to Bugzilla at any time. These bug reports are used to guide corrective maintenance activities and improve software quality. Free-form text fields of bug reports include title and description. Bugzilla also allows developers to track status and comment of bug reports and submit related attachments such as screenshots.

Since bug triage takes up a large amount of developer resources, many software projects have a shortage of them for dealing with all bug reports. In addition, a significant amount of submitted bug reports are duplicates that describe already-reported defects [6]. Previous studies indicate that more than 36 percent of bug reports are duplicates or invalid [7]. Generally, developers spend much time for identifying duplicates from a great number of bug reports in bug tracking system. Thus, identifying duplicate bug reports has been an important part of software maintenance.

Therefore, in this study we propose a technique to classify bug reports for identifying duplicate reports. This classification technique based on bug rule, combines textual similarity[8] and taxonomy algorithm[9]. The classification involves identifying duplicates and invalid bug reports to give developers a better result of bug reports retrieval. In order to help developers to find appropriate bug reports, we apply developers' feedback[10][11] to refine the result of classified bug reports. In practice, giving feedback is a kind of filtering between developers and incoming bug reports. Developers can rate bug reports as duplicate or not, and valid or not. In addition, they can verify whether bug reports belong to a specific bug type.

We use bug reports crawled from Bugzilla for experiment and classify them by applying bug rules. The experiment shows that this technique is effective and accurate.

The main contributions of this study are:

- By utilizing texture similarity and taxonomy algorithm, the bug reports from Bugzilla are classified according to bug rules. Developers do not spend much time in finding appropriate bug reports by efficiently identifying duplicates.
- By utilizing developers' feedback, they are allowed to verify which bug report is duplicate or not, and valid or not. In addition, they can justify if a bug report belong to a specific bug type. Giving feedback is a valid way for improving accuracy of bug report retrieval.

The paper is organized as follows. Section Ⅱ presents related works. In Section Ⅲ, we describe a data model of bug report in our classification technique. Section Ⅳ provides details of textural similarity and taxonomy algorithm. In addition, how to execute feedback mechanism is described in this section. We show our experiment in Section Ⅴ. Section Ⅵ summarizes our work and presents future works.

II. RELATED WORK

Runeson et al. proposed an approach to identify duplicate bug reports by using Natural Language Processing (NLP) techniques [12]. They developed a prototype tool for evaluating and analyzing bug reports. The evaluation shows

* Corresponding author

978-0-7695-4388-8/11 $26.00 © 2011 IEEE
DOI 10.1109/CIT.2011.90

that about two-thirds of the duplicates can be found possibly by using the NLP techniques. Since these bug reports are not classified in their work, it is not easy to find related bug reports. Moreover, if some reports among duplicates which the prototype found are invalid, developers spend much time in finding appropriate valid bug reports. Our work resolves these problems by applying a bug rule based classification technique and giving developers' feedback.

Some web-based tracking systems have been used to provide useful troubleshooting advice. However, it is still difficult to process semi-structured data in bug tracking systems. H. M. Tran et al. proposed a semantics-based bug search system[13] for extracting semi-structured data from other bug tracking systems. They described a unified data model to store bug tracking data. In their work, the multi-vector representation method (MVR) has been used to perform semantic search on the unified bug dataset. The work showed that the combination of full text search and meta-data search outperforms the other combinations. However, the work does not consider developer's perspective and feedback.

N. Jalbert and W. Weimer proposed a system that automatically classifies duplicate bug reports for saving developers' time [3]. They applied surface feature, texture semantics, and graph clustering to detect duplicate bug reports. They showed that the system reduced development cost by filtering out 8 percent of duplicate bug reports. Their study described that inverse document frequency was not useful in this task and title similarity was the most useful element for detecting duplicate bug reports. However, the work does not recognize that experienced developer' knowledge is also very effective for classifying bug reports.

While many detecting techniques of duplicate bug reports are exploiting only natural language information to implement the detecting task, X. Wang et al. present a new method which combines natural language information and execution information [14]. When a new bug report is submitted, the method retrieves the most similar bug reports to the new bug report by comparing natural language information and execution information of the report with those of existing bug reports. Then developers examine the suggested bug reports to determine whether the new bug report duplicates an existing bug report. They showed that the method detected from sixty-seven to ninety three percentage of duplicate bug reports in the Firefox bug repository. The number of detected bug reports is more than that of the method which uses only natural language information. However, even if execution information helps the method to improve the accuracy of detecting duplicate bug reports, it leads to increase in workload of the method.

III. MODEL FOR BUG REPORTS

In this section, we present a model for bug reports to use in this technique. The model shows how to structure bug reports to retrieve appropriate bug reports and give feedback.

The formal model represents an infra-structure of our bug reports classification and identification technique. The model uses some features extracted from bug reports to compute textural similarity, classify bug reports and implement feedback mechanism.

Fig. 1 shows the formal model. This model includes concept, contents, context and category properties. Thus we call the model "4C model".

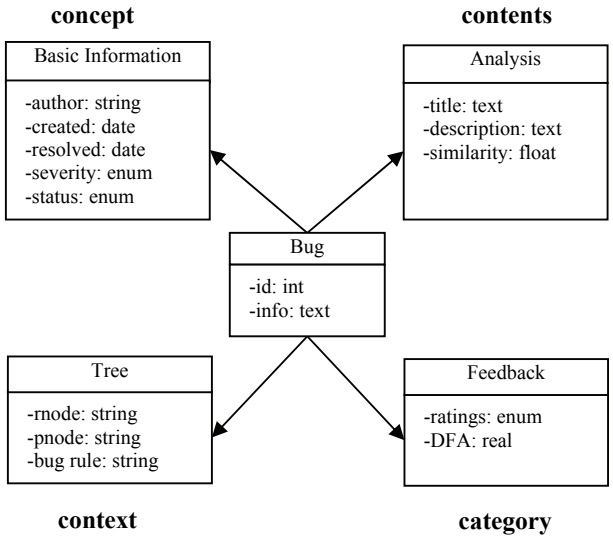

Figure 1. 4C model for bug reports

In 4C model, ***Bug***, with id and information, is based on four properties. The concept property indicates the basic information of bug report such as author, created date and so on. Specifically, the severity attribute represents the urgency level of bugs. It has the value of *critical*, *normal* or *minor*. The status attribute has two values: the value *open* indicates an unconfirmed, new, assigned and reopened bug in the bug tracking systems while the value *fixed* represents a resolved, verified and closed bug. The contents property shows detailed information of bug reports which include title, description and similarity. Attribute similarity represents a similarity measure value between bug reports in the database. The context property indicates a taxonomy tree structure which includes root node and parent node. This taxonomy tree is produced by applying taxonomy algorithm to bug reports. In addition, it includes bug rule information. The category property represents rating states of developers' feedback and a computed score using feedbacks, where each state is expressed as a number.

4C model implicitly shows a general outline of bug rule based classification technique with feedback. The model supports the core classification and feedback process. By applying textural similarity, taxonomy algorithm and feedback mechanism, bug reports are classified. Furthermore, duplicate and invalid bug reports are identified.

IV. BUG RULE BASED CLASSIFICATION TECHNIQUE WITH FEEDBACK

This section describes a way to classify bug reports, and identify duplicate and invalid bug reports. First, we describe how to compute a similarity measure between bug reports. Next, we show the bug reports classification process by applying taxonomy algorithm. Finally, we present a feedback mechanism.

A. Textural analysis and clustering

In order to identify duplicate bug reports, we need to measure a similarity between two bug reports. In this study, title and description of bug reports are distinguished and separate similarities are computed because the relative importance of title and description is different. Once we have obtained the vector v_i and v_j for elements e_i and e_j (title or description), we compute their similarity by using cosine algorithm[15].

Definition 1: Textural Similarity

$$\cos(\mathrm{v_i}, \mathrm{v_j}) = \frac{\mathrm{v_i} \cdot \mathrm{v_j}}{|\mathrm{v_i}| \times |\mathrm{v_j}|} = \frac{\sum_{k=1}^{n} w_{kj} \times w_{kj}}{\sqrt{\sum_{k=1}^{n} w_{ki}^2} \times \sqrt{\sum_{k=1}^{n} w_{kj}^2}}$$

where

- w_{ki} is the weight of k-th word in element e_i.
- The size of the word set is n.
- The value of weight (w_{ki} or w_{kj}) is computed by TF-IDF:

$$w_{ki} = tf_{ki} \times \log \frac{N}{n_k}$$

- tf_{ki} is the frequency of k-th word in element e_i.
- N is the total number of elements, and n_k represents the number of elements where k-th word occurs at least one.

After we compute similarity measures between titles and descriptions separately, we compute a similarity between two bug reports.

Definition 2: Similarity Between Bug Reports (SBBR)

$$\mathrm{SBBR}(\mathrm{b_i}, \mathrm{b_j}) = \alpha \times ts_{ij} + (1 - \alpha) \times ds_{ij}$$

where

- ts_{ij} is the textural similarity value between titles of bug report b_i and b_j.
- ds_{ij} represents the similarity value between descriptions of bug report b_i and b_j.
- α indicates the relative weight of title in full bug reports.

Next, we use textural similarity measures to group the bug reports by applying the hierarchical clustering algorithm[16]. Fig. 2 describes the clustering process of bug reports when a bug report is a cluster initially. When the distance, the reciprocal of similarity, between cluster c_α and c_β is a minimum, in other words, they have the highest similarity and the distance is smaller than a threshold, c_α and c_β are grouped into a cluster c_{new} and the number of clusters decreases by one. The minimum distance between clusters c_k and c_α or c_k and c_β is the distance between c_k and c_{new}. Finally, we obtain a tree of bug reports. The time complexity of this algorithm is $O(N^3)$.

In order to verify duplicate bug reports accurately, we need to set a cutoff value for choosing a part of the hierarchical tree as the targets of classification. Nodes with the similarity larger than the cutoff value are classified by our taxonomy algorithm. We describe the cutoff value in Section Ⅴ.

```
//Hierarchical Clustering Algorithm (HCA)
Begin
  input all of bug reports;
  make each bug report b_i into a cluster c_i ; //1 ≤ i ≤ N, N is the number
  of bug reports.
  C = { c1, c2, …, cN };
  set a cutoff value to v;
  while(true) {
    find c_α and c_β such that d [(c_α), (c_β)] = min (d [(c_i), (c_j)] ), for 1 ≤ i,
    j ≤ N and i ≠ j; // d represents the distance between bug reports,
    while the distance is computed by the reciprocal of similarity
    measure.
    if (v < d [(c_α), (c_β)]) break;
    c_new = c_α ∪ c_β;
    C = C – { c_α, c_β };
    C = C ∪ {c_new};
    d[(c_k), (c_new)] = min (d[(c_k), (c_α)], d[(c_k), (c_β)]); // c_k is other cluster.
  }
End
```

Figure 2. Bug reports clustering

B. Classification

In order to help developers to find related bug reports easily and quickly for debugging defects conveniently, it is necessary to classify bug reports after clustering process.

Before we introduce our bug reports classification mechanism, we show four bug types as follows:

- *Type 1. The range of array is out of bounds.*
- *Type 2. The variable is not initialized: the variable need to be initialized after definition. If the variable is not initialized, the value cannot be fixed so that the result is unknown.*
- *Type 3. Non executable node: Because of some known and unknown reasons, a segment code does not just run by itself.*
- *Type 4. Some methods such as equals(), hashCode() or clone( ) cannot be used exactly.*

The bug reports with the similarity larger than the cutoff value are selected according to these bug types. We apply a popular taxonomy algorithm to classify these bug reports.

Taxonomy is the practical science of classification. A taxonomy, or taxonomy scheme [17], is a particular classification, while elements arrange in a hierarchical tree. The tree structure is organized by supertype-subtype relationships or parent-child relationships. Fig. 3 shows an example of taxonomy according to four types. By scanning keywords of bug types, we verify which bug class is arranged to include related bug types. In this way, a taxonomy tree is constructed. When clustering process is completed, bug reports are classified according to related bug rule. We give a definition of bug rule as follows:

Definition 3: Bug Rule(BR)

A bug is a defect in a software program which may lead to project failure. Generally, bugs are produced in the source code file. The bug report is a medium for reporting the bug. Bug rules are summarized by the history of existing bug reports in the database. Bug rules indicate that bugs should belong to the appropriate bug types. A bug rule represents a mapping relationship between a grammatical rule and a bug type while a grammatical rule indicates regulations with which developers must conform in programming.

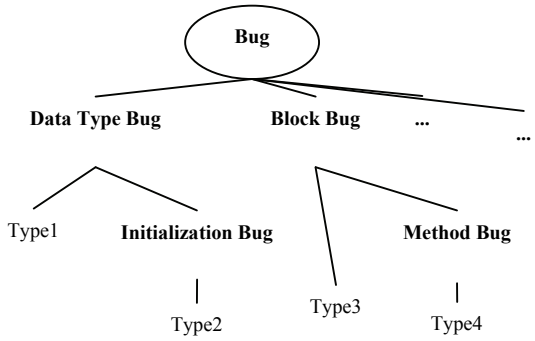

Figure 3. An example of taxonomy tree

According to bug rules, bug reports are classified into appropriate bug types. The process is executed by computing a similarity measure between a bug report and a bug rule. We define the measure as follows:

Definition 4: Similarity Between Bug Report and Bug Rule (SBRR)

$$\mathrm{SBRR}(b_i, r_j) = \beta \times \mathrm{tgs}_{ij} + (1-\beta) \times \mathrm{dts}_{ij}$$

where

- b_i is a bug report and r_j is a bug rule.
- tgs_{ij} indicates the similarity value between the title of a bug report b_i and the grammatical rule of a bug rule r_j.
- dts_{ij} represents the similarity value between the description of a bug report b_i and the bug type of a bug rule r_j.
- β is the relative weight of the title and the grammatical rule in full bug report and bug rule.

We set a threshold value to decide whether bug reports are classified into related bug types. When a similarity of a bug report is larger than the threshold value, the bug report is arranged to a corresponding bug type. In order to compute the similarity accurately between bug reports and bug rules, we compute the similarity between the title of a bug report and the grammatical rule of a bug rule, and the similarity between the description of the bug report and the bug type of the bug rule.

Now we describe the details of the process by using an example. First, we assume that we selected a new bug report from Bugzilla as follows:

<Bug #484206>

Title: Software error when granting a user access to a large number of groups.

Description: When I grant a user access to a large number of groups, Bugzilla throws an error. I suspect this is due to the INSERT INTO profiles_activity action online 315 of editusers.cgi. If the string of group names becomes long, it no longer fits into the 255 character limit of the new value column of profiles_activity.

By using the similarity measures between the bug report and the bug rules, we found that the bug rule "limited size of group={The range of array is out of bounds}" is the most similar rule to ***Bug #484206*** because the similarity measure is 0.85. Thus, ***Bug #484206*** is arranged to ***Type1*** in the taxonomy tree.

It is noteworthy that if the similarity measures between a bug report and more than one bug rule are larger than the threshold value, the bug report are arranged to more than one bug types in the taxonomy tree. Thus it is necessary to utilize developer feedback to verify the arrangements of the bug report.

C. Developer feedback

A feedback mechanism plays an important role in this bug report classification. By utilizing developer's feedback, we improve the accuracy of bug report classification. In our work, developer's feedback has the following functions:

- Duplicates identification: Even if our texture analysis and clustering approach are used to find duplicate bug reports, the accuracy may not be satisfactory. The feedback allows developers to decide whether a bug report is a duplicate or not.
- Validity identification: We are able to justify whether a bug report is valid or not by analyzing developers' feedback rating data.
- Classification identification: Though our taxonomy algorithm is used to classify bug reports, some bug reports may be arranged to improper bug types. Developer's feedback helps to decide whether the bug report belongs to the right type in the taxonomy tree by the rating process.

We define developer feedback algorithm to quantify the rating process.

Definition5: Developer Feedback Algorithm (DFA)

$$\mathrm{DFA}(b) = \frac{\sum_{i=1}^{n} S(d_i, b)}{\sum_{j=1}^{m}\sum_{i=1}^{n} S(d_i, b_j)}$$

where

- $S(d_i, b)$ shows a score that developer d_i gives bug report b after the classification process.
- n is the number of developers.
- m is the number of bug reports in the same bug type.

This technique computes a *DFA* of a bug report and removes the bug report if the DFA is smaller than a predefined threshold.

V. EXPERIMENT

In this section, we demonstrate the practical feasibility of the bug rule based bug reports classification technique with feedback. First, we collect bug reports from Bugzilla. Next, we present the feasibility of this classification technique by a case study. Finally, we show comparisons with other studies.

A. Crawling the bug reports from Bugzilla

The Bugzilla bug tracking system provides an XML-RPC web service interface for developers to access and modify bug reports. In addition, the Bugzilla XML-RPC API allows programs to retrieve a large number of bug reports by entering the keywords as a query request. It saves much time when programs download many bug reports from a single Bugzilla website that supports XML-RPC.

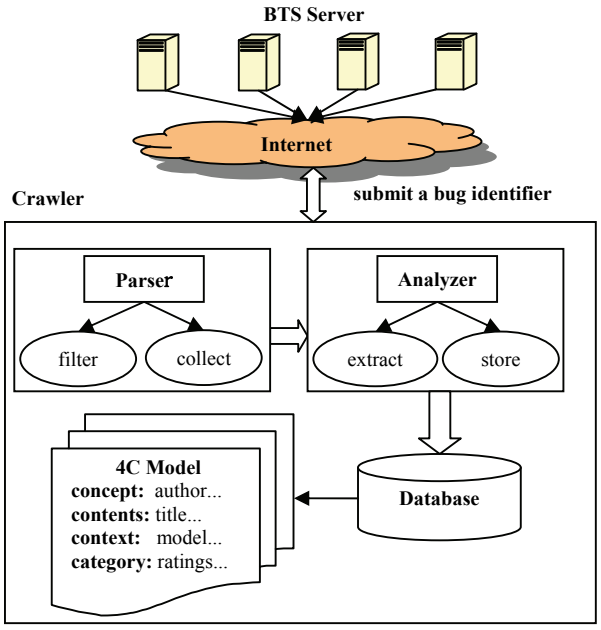

Figure 4. Architecture of the crawler tool

In order to gather bug reports from Bugzilla successfully, we developed a crawler tool for parsing HTML pages from Bugzilla server. Fig. 4 shows the architecture of the crawler tool. The role of the crawler tool is an XML-RPC client that submits a bug identifier (keyword) to a Bugzilla server and gets the details of related bug reports from the Bugzilla server. The tool includes Parser and Analyzer. Parser filters the HTML tags and collects the textual contents and Analyzer extracts and stores related features which correspond to the attributes of 4C model. In this experiment, we only collect bug reports which are related to four bug types that we described.

B. A case study

By utilizing bug reports crawler we developed, some bug reports are collected into the database. In order to debug these errors easily, developers need to know which bug reports are duplicates or not and which type the bug reports belong to. Through computing textural similarity measures and executing hierarchical clustering algorithm and taxonomy classification strategy, we get the result of bug report classification. Fig. 5 shows a screenshot of the classification process. We assume that α and β in Definition 2 and 4 are 0.75 and 0.65 in case study, respectively.

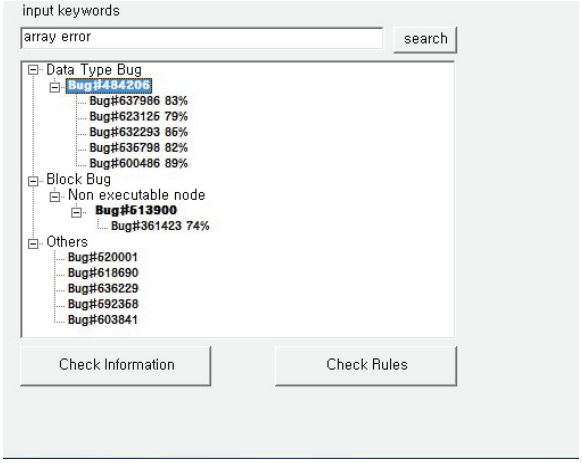

Figure 5. Bug reports classification process

Fig. 5 shows that the tool displays a tree structure when a developer enters a keyword 'array error' and click the button ***search***. Once a developer submits a query request, the tool computes the similarity measure between the keyword and the bug reports in the database by applying Definition 1. If the similarity measure is larger than a threshold, the bug report and its duplicates are shown in the taxonomy tree. In this case study, we assume that the threshold is 0.75. Fig. 5 shows that the original bug report ***Bug #484206*** and ***Bug #613900*** are arranged to ***Type 1(The range of array is out of bounds***) and ***Type 3(Non executable node)***, respectively. Non-bold child nodes represent their duplicates, and subsequent percentage value represents the similarity measure between the original bug report and duplicate bug report. We also assume that a cutoff value for selecting the candidates of classification is 0.7. In addition, if the similarity measures between the bug reports and appropriate bug rules are smaller than the threshold, these bug reports are classified into ***Others***.

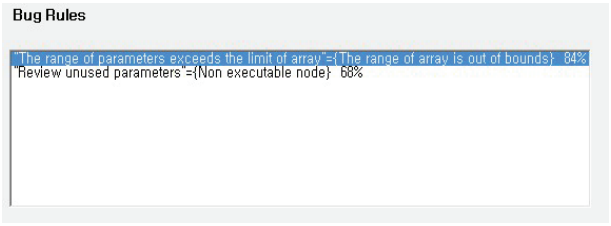

Figure 6. Bug rules of Bug #484206

Developers check appropriate bug rules of the original bug report by clicking the button ***Check Rules*** of the tool. Fig. 6 shows two bug rules of ***Bug #484206***. The left side of equal sign in a bug rule is a grammatical rule. The right side is a

related bug type which we defined. Bug rules are defined to decide which bug type the bug report belongs to. Subsequent percentage value represents the similarity measure between the original bug report and an appropriate bug rule. If the similarity measure is larger than the threshold, the corresponding original bug report is arranged to relevant bug type. In this experiment, we assume that the threshold is 0.8. Developers click the button ***Check Information*** to look for the details of a bug report in the taxonomy tree. Fig. 7 shows the details of ***Bug #484206***.

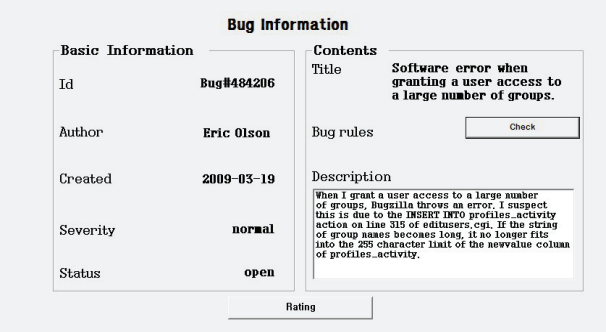

Figure 7. Details of Bug #484206

According to 4C model, bug information includes basic information and contents. These features in Fig. 7 correspond to the properties in 4C model. Developers can click the button ***Check*** to look up related bug rules and click the button ***Rating*** to score this bug report.

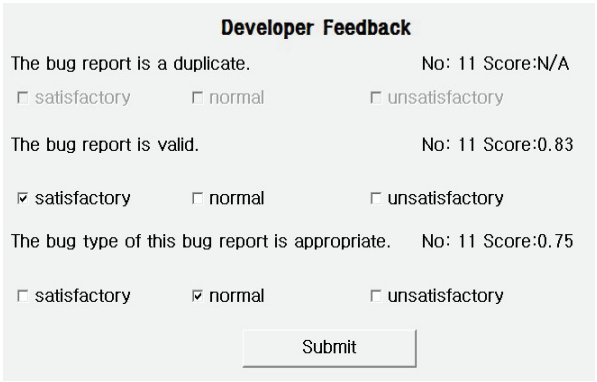

Figure 8. Developer feedback process

Developers can score this bug report by choosing an appropriate evaluation. Fig. 8 shows the developer feedback process of ***Bug #484206***. The developer has to choose an evaluation which corresponds to a feedback item. To quantify an evaluation in the rating process, we use 0, 1 and 2 numeric values to represent three states "unsatisfactory", "normal" and "satisfactory" of evaluations. Developers choose an appropriate state for identifying a duplicate, invalid bug report and a related bug type. The first feedback item and a corresponding evaluation are unused because this bug report is original and it is not necessary to be identified whether the bug report is a duplicate or not. Specially, *No* is the number of developers who rate the bug report and *Score* represents a feedback score by applying *DFA* described in Definition 5**.** After rating the bug report, a developer can click the button ***Submit*** to upload the feedback information. Then our tool recomputes the feedback score. If the score is smaller than the threshold (we assume that the threshold is 0.75), the bug report is arranged to ***Others*** in the taxonomy tree.

C. Performance

In this section, we evaluate the performance of the classification technique. In order to compare this technique with other classification cases, we define *Precision-Recall*[18][19] as performance measure. *Precision* indicates the number of correct retrieved bug reports divided by the total number of retrieved bug reports. *Recall* represents the number of correct retrieved bug reports divided by the total number of correct bug reports.

Definition6: Precision-Recall Measure

$$\text{Precision}(q) = \frac{|C(r,q) \cap R(r,q)|}{|R(r,q)|}$$

$$\text{Recall}(q) = \frac{|C(r,q) \cap R(r,q)|}{|C(r,q)|}$$

where

- $C(r, q)$ indicates a set of correct bug reports identified by experienced developers such as d_1, d_2, etc with respect to a query q.

$$C(r,q) = C(d_1,r,q) \cap C(d_2,r,q) \cap \cdots \cap C(d_n,r,q)$$

- $R(r, q)$ represents all bug reports retrieved by a bug tracking system with respect to a query q.
- n is the number of experienced developers who participate in the performance assessment.

We compute the precision-recall value of the following three cases in this assessment:

- Bug reports retrieval in Bugzilla: In Bugzilla, developers enter the keywords to get related bug reports.
- Bug reports retrieval without feedback in our technique: In our tool, developers enter keywords to get related bug reports, but feedback process is not applied.
- Bug reports retrieval with feedback in our technique: In our tool, developers enter keywords to get related bug reports and our bug rule based bug reports classification technique with feedback is performed completely.

Fig. 9 demonstrates the evaluation result of three cases by using Precision-Recall measure. The data on X-axis indicates the value of recall, and the data on Y-axis represents the value of precision. From left to right side of the line, six data points represent the number of retrieved bug reports from 10 to 35 while the step length is 5. We can see that the more bug reports are retrieved, the better the recall and the worse the precision is.

Figure 9. Comparison of three cases

The uppermost line shows the best precision and recall, while the performance of this classification technique with feedback outperforms other cases. With the classification and feedback process of this technique, the precision reaches 0.9 when 10 bug reports are retrieved in experiments. If the feedback process is not applied, the precision reaches 0.8. Since the average difference is around 10 percent, the feedback process is important for improving the accuracy of bug reports retrieval. Bugzilla has the worst precision and recall among three cases. In addition, with the increasing number of retrieved bug reports, the precision decrease of this technique with feedback is not radical than other cases. Finally, by utilizing the classification method with feedback, developers are expected to get better result of bug reports retrieval.

D. Discussion

As bug reports retrieval has been a hot spot in software maintenance, many bug tracking systems have been developed for assisting developers. However, if classification and feedback process are not applied, developers cannot get favorable result. From the previous section, we can see that the classification and feedback process are essential because they help to enhance the accuracy of bug reports retrieval so that developers are able to debug errors effectively.

Table 1 shows the comparisons of this study and other bug reports retrieval studies. Other works are Bugzilla [4][5], the semantics-based bug search study[13] and the bug retrieval system which combines natural language and execution information(N&E) [14].

From Table Ⅰ, we can see that all studies have clustering process because clustering is a necessary step for retrieving bug reports. The comparison shows that Bugzilla and semantic-based bug search do not classify bugs, while this study and N&E study classify them. None of these studies supports ranking process of bugs. However, ranking is not necessary for this study and N&E study because these studies support feedback. The feedback process of this study is different from that of N&E work. This study includes the duplicates identification, validity identification and classification identification while N&E work only includes the duplicate identification. Therefore, this classification technique is expected to improve the accuracy of bug report retrieval.

TABLE I. COMPARISON WITH BUG RETRIEVAL SYSTEMS

Criteria	Bug retrieval studies			
	[4][5]	[13]	[14]	This work
clustering	*Yes*	*Yes*	*Yes*	*Yes*
classification	*No*	*No*	*Yes*	*Yes*
ranking	*No*	*No*	*N/A*	*N/A*
feedback	*No*	*No*	*Yes*	*Yes*

VI. CONCLUSION

In this paper, we proposed a bug rule based bug report classification technique with feedback for duplicate identification and bug reports retrieval. By performing textual analysis, clustering and classification, the bug reports which are crawled in Bugzilla have been classified into a taxonomy structure according to 4C model. In order to improve the accuracy of duplicate identification and bug report retrieval, we applied the feedback process to rate these bug reports. Developers decide whether a bug report is a duplicate, the report is valid and it belongs to an appropriate bug category in the feedback process.

Though this bug report classification has been employed effectively, the number and types of bug reports crawled from Bugzilla have been limited.

In the future, we plan to automatically execute feedback process by utilizing the expert system or collective intelligence technique, where developers need not to do some tasks for rating bug reports. It may cut down the cost of this technique. We will also extend bug types to classify more bug reports.

VII. ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (No. 2010-0025477 and No. 2010-0028148).

REFERENCES

[1] N. Ramasubbu and R. K. Balen, "Globally Distributed Software Development Project Performance: an Empirical Analysis," Proc. of the 6th joint meeting of the European software engineering conference, 2007, pp. 125-134.

[2] J. Anvik and G. C. Murphy, "Determining Implementation Expertise from Bug Reports," Proc. of the Fourth International Workshop on Mning Software Repositories, 2007, pp. 1-8.

[3] N. Jalbert and W. Weimer, "Automated Duplicate Detection for Bug Tracking System," Proc. of International Confterence on Dependable System & Networks, 2008, pp. 52-61.

[4] The Bugzilla website : http://www.bugzilla.org/

[5] J. Remillard, "Source Code Review System," IEEE Software, 2005, Vol. 22 (1), pp. 74-77.

[6] C. Sun, D. Lo, Xiaoyin. W, J. Jiang and S. Khoo, "A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval," Proc. of the 32nd ACM/IEEE International Conference on Software Engineering, 2010, vol.1, pp. 46-54.

[7] J. Anvik, L. Hiew and G. C. Murphy, "Who Should Fix This Bug?" Proc. of the 28th International Conference on Software Engineering, 2006, pp. 361-370.

[8] A. Islam and D. Inkpen, "Semantic Text Similarity using Corpus-based Word Similarity and String Similarity," ACM Transactions on Knowledge Discovery from Data, 2008, vol. 2(2), pp. 1001-1025.

[9] The WIKIPADIA wibesite [Online].
http://en.wikipedia.org/wiki/Taxonomy

[10] T. Zhang, B. Lee, H. Kim, J. Lee, S. Kang and I. Shin, "Semantic Concept based Retrieval of Software Bug Report with Feedback," Proc. of International Conference on Internet and Multimedia Technologies, 2010, vol.3, pp.51- 55.

[11] E. Agichein, E. Brill and S. Dumais, "Improving Web Search Ranking by Incorporating User Behavior Information," Proc. of the 29th International ACM SIGIR Confterence on Research and Development in Information Retrieval, 2006, pp.19-26.

[12] P. Runson, M. Alexandersson and O. Nyholm, "Detection of Duplicate Defect Reports Using Natual Language Process," Proc. of 29th International Conference on Software Engineering, 2007.

[13] H. M. Tran, G. Chulkov and J. Schönwälder, "Crawling Bug Tracker for Semantic Bug Search," Proc. of IFIP International Federation for Information Processing, 2008, pp. 55-68.

[14] X. Wang, L. Zhang, T. Xie, J. Anvik and J. Sun, "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution," Proc. of ACM/IEEE International Conference on Software Engineering, 2008, pp. 461-470.

[15] C. H. Brooks and N. Montanez, "Improved Annotation of the Blogosphere via Autotagging and Hierarchical Clustering," Proc. of the 15th International Conference on World Wide Web, 2006, pp. 625-631.

[16] H. Hastie, R. Tibshirani and J. Friedman, "Hierarchical clastering," The Elements of Statistical Learning, New York: Springer Inc, 2001, pp. 272-280.

[17] C. Marlow, M. Naaman, D. Boyd and M. Davis, "HT06, Tagging Paper, Taxonomy, Flickr, Academic Article, to Read," Proc. of the Seventeenth Conference on Hypertext and Hypermedia, 2006, pp.31-39.

[18] J. Davis and M. Goadrich, "The Relationship between Precision-Recall and ROC Curves," Proc. of the 23rd International Conference on Machine Learning, 2006, pp.233-240.

[19] O. Chum, J. Philbin, J. Sivic, M. Isard and A. Zisserman, "Total Recall: Automatic Query Expansion with a Generative Feature Model for Object Retrieval," Proc. of IEEE 11th International Conference on Computer Vision, 2007, pp. 1-8.