PREDICTING SOFTWARE BUGS USING WEB LOG ANALYSIS TECHNIQUES AND NAÏVE BAYESIAN TECHNIQUE

D.PADMABHUSHANA[1], D.SRIKANTH[2]

DEPARTMENT OF CSE,

Gudlavalleru Engineering College, Gudlavalleru

Abstract - **With the continued growth and proliferation of Web services, software as a services and Web based information systems, the volumes of user data have reached astronomical proportions. As the World Wide Web is continuously and rapidly growing, it is necessary for the web miners to utilize intelligent tools in order to find, extract, filter and evaluate the bugs information. The data pre-processing stage is the most important phase for investigation of the web bugs behavior. To do this one must extract the only human user accesses from weblog data which is critical and complex. Hence an extensive learning algorithm is required in order to get the desired information. Software defect (bug) repositories are great source of knowledge. Data mining can be applied on bug repositories to explore useful interesting patterns. Complexity of a bug helps the development team to plan future software build and releases. This paper introduces an extensive research frame work capable of pre processing web log bug data completely and efficiently. The framework reduces the error rate and improves significant learning performance of the algorithm. This framework helps to investigate the software bug behavior efficiently. For this Naïve Bayesian classifier is applied to predict for the future depending on the current analysis outcomes. Our system is intended to provide for Web Site Maintainers, Web Site Developers, Personalization Systems, Pre-fetched Systems, Recommender Systems and Web Site Analysts as well as software developers to analyze the bugs in the software code.**

Keywords - **Web server log, Web usage mining, Data mining, User access patterns.**

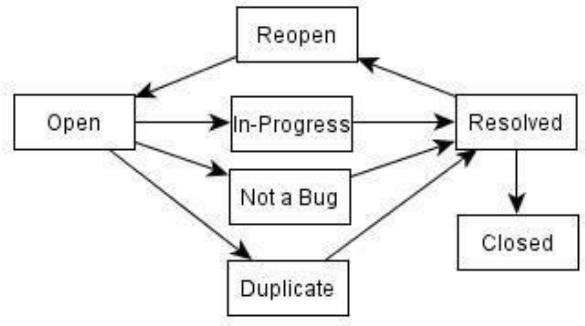
I. INTRODUCTION

World Wide Web (WWW) is very popular and interactive. It has become an important source of information and services. The web is huge, diverse and dynamic. Extraction of interesting information from Web data has become more popular and as a result of that web mining has attracted lot of attention in recent time [1]. Web mining is an application of data mining to large web data repositories [2].It can be divided in to three categories namely web structure mining, web content mining and web usage mining. Web mining is the automatic discovery of user access patterns from web servers. Web usage mining is an important technology for understanding user's behaviors on the web and is one of the favorite area of many researchers in the recent time. Obtained user access patterns can be used in variety of applications, for example, one can keep track of previously accessed pages of a user. These pages can be used to identify the typical behavior of the user and to make prediction about desired pages [4]. Thus personalization for a user can be achieved through web usage mining. Mass customization and personalization performed by dynamic.

Many commercial web log analyzer tools are available in the market that analyzes the web server log data to produce different kinds of statistics. In this study, web log expert program has been used to analyze server log data of a website. Program generated different types of reports on server log data that can be useful from the point view of system administrator or web designer to increase effectiveness of the site. It is important to note that preprocessing is a necessary step in web usage mining before applying any technique on usage data to discover user access patterns. As far as mining of knowledge from the data is concerned, quality of data is a key issue. Nearly 80% of mining efforts often spend to improve the quality of data . The data which is obtained from the logs may be incomplete, noisy and inconsistent. The attributes that for in quality data includes accuracy, completeness, consistency, timeliness, believability, interpretability and accessibility. So preprocessing of data is required to make it have the above mentioned attributes and to make it easier for mining.

A bug is defect in sofware. Bug indicates the unexpected behavior of some of the given requirement during software development. During software testing the unexpected behavior of requirements are identified by software testers or quality engineers and they are marked as a Bug. In this paper both defect and bug are used as synonyms. Bugs are managed and tracked using number of available tools like Bugzilla, Perforce, JIRA etc. Most of the open source projects and large projects manages their software development related data using some of the project

management tools. For managing the bugs associated with the software bug tracking tools are used. These bug tracking systems provides online interfaces to various users associated with the projects. These tools internally manages the bug repositories where all the bugs and related data are stored. For example for the Mozilla project, the bugs are tracked using bugzilla tool [10]. Bugzilla provides all the mozilla bugs in the form of online repository. By specifying the bug id in the Mozilla' online repository, any user can fetch the required bug information. The url for Mozilla's bug repository is"https://bugzilla.mozilla.org/show_bug.cgi?id=". A software bug enters into the various state for its resolution. Figure-1 depicts a general bug state diagram. The boxes indicates various bug states and arrow indicates the tranisition between the states. The most common and simple path which a bug follow is Open → In-Progress → Resolved → Closed. When a bug is identified by a tester or by a quality engineer its summary, description and related informations are entered into the bug tracking system and during this action item every bug gets one uniqe id number. As soon as the bug is created it enteres into the "Open" state. And it is assigned to one of the developer for fixation. Once it is assigned to a developer and he or she start working for the resolution, the bug enteres into the "In-Progress" state, after fixation of the bug developer mark that bug as "Resolved", which is the "Resolved" state and it is assigned back to the tester or quality engineer for verification. Once the bug is verified by tester or quality engineer and found ok then it is marked as "Closed".



II. RELATED WORK

A software bug is what software engineers commonly use to describe the occurrence of a fault in a software system. A fault is then defined as a mistake which causes the software to behave differently from its specifications. Nowadays, users of software systems are encouraged to report the bugs they encounter, using bug tracking systems such as Jira [www.atlassian.com/software/jira] or Bugzilla [www. bugzilla.org]. Subsequently, the development is able to make an effort to resolve these issues in future releases of their applications. Bug reports exchange information between the users of a software project experiencing bugs and the developers correcting these faults. Such a report includes a one-line summary of the observed malfunction and a longer more profound description, which may for instance include a stack trace. Typically the reporter also adds information about the particular product and component of the faulty software system: e.g., in the GNOME project, "Mailer" and "Calendar" are components of the product "Evolution" which is an application integrating mail, address-book and calendaring functionality to the users of GNOME. Researchers examined bug reports closely looking for the typical characteristics of "good" reports, i.e., the ones providing sufficient information for the developers to be considered useful. This would in turn lead to an earlier fix of the reported bugs [8]. In this study, they concluded that developers consider information like "stack traces" and "steps to reproduce" most useful. Since this is fairly technical information to provide, there is unfortunately little knowledge on whether users submitting bug reports are capable to do so. Nevertheless, we can make some educated assumptions. Users of technical software such as Eclipse and GNOME typically have more knowledge about software development, hence they are more likely to provide the necessary technical detail. Also, a user base which is heavily attached to the software system is more likely to help the developers by writing detailed bug reports.

III. PROPOSED APPROACH

The primary steps involved in performing change classification on a single project are outlined as follows:
Creating a corpus:
1. File level changes are extracted from the revision history of a project, as stored in its SCM repository.
2. The bug fix changes for each file are identified by examining keywords in SCM change log messages
3. The bug-introducing and clean changes at the file level are identified by tracing backward in the revision history from bug fix changes 4. Features are extracted from all changes, both buggy and clean. Features include all terms in the complete source code, the lines modified in each change (delta), and change metadata such as author and change time. Complexity metrics, if available, are used at this step. The following step is the new contribution in this paper.

Feature Selection:

5. Perform a feature selection process that employs Gain Ratio to compute a reduced set of features. For each iteration of feature selection, classifier performance is optimized for a metric (typically F-measure or accuracy). Feature selection is iteratively performed until optimum points are reached. At the end of Step 5, there is a reduced feature set that performs optimally for the chosen classifier metric.

Classification:

6. Using the reduced feature set, a classification model is trained. Although many classification techniques could be employed, this paper focuses on the use of Na¨ıve Bayes .
7. Once a classifier has been trained, it is ready to use. New changes can now be fed to the classifier, which determines whether a new change is more similar to a buggy change or a clean change.

A. **Finding Buggy and Clean Changes**

In order to find bug-introducing changes, bug fixes must first be identified by mining change log messages. We use two approaches: searching for keywords in log messages such as "Fixed", "Bug", or other keywords likely to appear in a bug fix and searching for references to bug reports like "#42233". This allows us to identify whether an entire code change transaction contains a bug fix. If it does, we then need to identify the specific file change that introduced the bug. For the systems studied in this paper, we manually verified that the identified fix commits were, indeed, bug fixes. For JCP, all bug fixes were identified using a source code to bug tracking system hook. As a result, we did not have to rely on change log messages for JCP.

Feature Extraction

A file change involves two source code revisions (an old revision and a new revision) and a change delta that records the added code (added delta) and the deleted code (deleted delta) between the two revisions. A file change has associated metadata, including the change log, author, and commit date. Every term in the source code, change delta, and change log texts is used as a feature. We gather eight features from change metadata: author, commit hour, commit day, cumulative change count, cumulative bug count, length of change log, changed LOC (added delta LOC + deleted delta LOC), and new revision source code LOC. We compute a range of traditional complexity metrics of the source code by using the Understand C/C++ and Java tools [6].

Preprocessing of Web Bug Data

In this step, we remove parts of the original web log data that are not relevant in our mining process. After that we get the web log data as 192.168.0.161 7/3/2008/12:00:05 http:// www.google.com/

- 192.168.0.161 is the IP address (client) that can be used to mind personal usage and the result can be applied in Personalized Systems, Recommender Systems and Pre-fetched System.
- . 7/3/2008 12:00:05 is the date time data that is intended to support the Web Site Maintainers, .Web Site Developers and Web Site Analyzer to know the most usage and the least usage date time. http://www.google.com/ is the domain name. Our analyzer is to know which IP frequently uses which sites for which purposes.

D. Web Usage based Mining

Mining Specific Site – to support for Web Site Maintainers, Web Site Developers and Web Site Analysts (Statisticians).

Mining Specific Client – to know the clients' frequent usage behavior, site and purposes. This mining result can be especially applied in Personalized Systems. Mining Specific Purpose – Our system can analyze the most usage site for a particular purpose. From the Web Site Maintainers' and Web Site Developers' point of view, they can prepare to attract more and more clients for their sites. Mining Specific Day – to report which day is the peak usage day of the week. The result is most suitable for Web Site Analysts to view the statistics of web usage data. Web Site Developers and Maintainers can prepare/modified their Web Site structure to persuade more and more clients. Mining Frequent Item Set – Discovering, extracting and comparing the clients' usage, percentage of different search engines' usage, usage rate among date , day etc.

To fulfill our purposes, we propose our own procedures that may be useful for various domain areas.

Procedure 1: **PPR**. Mining Specific Client's Bug Usage
Input: Database D of transactions, Specific IP address
Output: Sites (S) used by individual user
Method
1. Accept input_IP (Specific IP Address)
2. for (int i=0; i<= D.size; i++)
3. if (input_IP = = D_IP)
4. extract sites from database
5. return S

For the Web Site Maintainers, they should know the users' interest rate on their sites. Depending on the users' interest rate decreasing or increasing through the time, they can modify their site structure to attract more users from the aspect of economic benefits. On the other hand, the statistician can view from the analysis perspective. We proposed the procedure AM especially for maintainers and developers as a result of Web usage over a specific period of time.

Procedure 2: Find the users' daily bug usage
Input: Database D of transactions, Specific Date for Web Sites usage
Output: total number of users in one day
1. Method
2. Accept date to count the daily Web usage
3. count= 0; temp[ ] =null; // initialization
4. while (accept_date= = D_date)
5. if(temp[ ] = =null)
6. temp[ ] = D_IP
7. count ++
8. else
9. while (temp [ ])
10. if (temp [ ]== D_IP)
11. do nothing
12. else temp[ ] = = D_IP
13. count ++
14. return count // total number of
15. internet users in one day

Procedure 3: Find the multi-users Software bug usage upon a specific period
Input: Database D of transactions, start_date and end_date to count the total number of usage during a specific period
Output: total number of users for a particular period
Method
1. Accept start_date and end_date
2. count = 0; temp [ ] = null;
3. while (D_date>= start_date && D_date<= end_date)

4. if (temp [ ] = = null)
5. temp[ ] = D_IP
6. count ++
7. else
8. while (temp [ ])
9. if(temp [ ] = = D_IP)
10. do nothing
11. else temp[ ]= D_IP
12. count ++
13. return count // total number of internet users
for a specific period

Procedure 4: Comparison of Open source software Bug Usage
Input: Database D of transactions, predefined rule table (T)
Output: Usage comparison and Percentage among search engine
Method
1. Extract keys from T where purpose= = 'Search'
2. initialize keys to zero, count =0;
3. for(int i=0; i<=D.size; i++)
4. if (D[i].domain= = key1)
5.key1++;. …..
6. else key n++;
7. count ++;
8. key1=key1/count*100;
9. …
10. key n=key n/count*100;
11. Return the usage percentage of search engine

2. BAYESIAN CLASSIFICATION

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities, such as the probability that a given tuple belongs to a particular class. Bayesian classification is based on Bayes theorem. Naïve Bayesian classifiers assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. It is made to simplify the computations involved and, in this sense, is considered "naïve." Bayesian belief networks are graphical models, which unlike naïve Bayesian classifiers, allow the representation of dependencies among subsets of attributes. Bayesian belief networks can also be used for classification.

Bayes' Theorem

Bayes' theorem is named after Thomas Bayes, a nonconformist English clergyman who did early work in probability and decision theory during the 18th century. Let X be a data tuple. In Bayesian terms, X is considered "evidence." As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis, such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine P(H/X), the probability that the hypothesis H holds given the "evidence" or observed data tuple X. In other words, we are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X. P(H/X) is the posterior probability, or a posteriori probability, of H conditioned on X. For example, suppose our world of data tuples is confined to customers described by the attributes age and income, respectively, and that X is a 35-year-old customer with an income of $40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then P(H/X) reflects the probability that customer X will buy a computer given that we know the customer's age and income. In contrast, P(H) is the prior probability, or a priori probability, of H. For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. The posterior probability, P(H/X), is based on more information (e.g., customer information) than the prior probability, P(H), which is independent of X. Similarly, P(X/H) is the posterior probability of X conditioned on H. That is, it is the probability that a customer, X, is 35 years old and earns $40,000, given that we know the customer will buy a computer. P(X) is the prior probability ofX.Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000. P(X/H), and P(X) may be estimated from the given data, as we shall see below. Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H/X), from P(H), P(X/H), and P(X). Bayes' theorem is

$$P(H|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|H)P(H)}{P(\boldsymbol{X})}.$$

Naïve Bayesian Classification

The naïve Bayesian classifier, or simple Bayesian classifier, works as follows:
1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2, : : : , xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2, : : : , An.
2. Suppose that there are m classes, C1, C2, : : : , Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if P(Ci/X) > P(Cj/X). Thus we maximize P(Ci/X). The classCi for which P(Ci/X) is maximized is called the maximum posteriori hypothesis. By Bayes' theorem

$$P(C_i|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|C_i)P(C_i)}{P(\boldsymbol{X})}.$$

3. As P(X) is constant for all classes, only P(X/Ci)P(Ci) need be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = …….. = P(Cm). Given data sets with many attributes, it would be extremely computationally expensive to compute P(X/Ci). In order to reduce computation in evaluating P(X/Ci), the naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes). Thus,

$$P(\boldsymbol{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i).$$

We can easily estimate the probabilities P(x1/Ci), P(x2/Ci), : : : , P(xn/Ci) fromthe training tuples. Recall that here xk refers to the value of attribute Ak for tuple X. For each attribute, we look at whether the attribute is categorical or continuous-valued.

IV. EXPERIMENTAL RESULTS

Online open source software bug usage Statistics about hits, page views, visitors and bandwidth are shown in table 1. Figure 2 shows the daily errors types. Different types of errors are shown in Table 2. It is clear from the table that 404 (Table 3) is most frequently occurred error. Some other types of client and server errors are shown in Table 4.

HITS	
Total Hits	30,474
Visitor Hits	29,191
Spider Hits	1,283
Average Hits Per Day	3,809
Average Hits Per Visitor	8.16
Cached Requests	3,979
Failed Requests	233
Page views	
Total Page Views	4,435
Average Page Views Per Day	554
Average Page Views Per Visitor	1.24
Visitors	
Total Visitors	3,576
Average Visitors Per Day	447
Total Unique IPs	3,038

Bandwidth	
Total Bandwidth	567.48 MB
Visitor Bandwidth	548.81 MB
Spider Bandwidth	18.67MB
Average Bandwidth Per Day	70.94MB
Average Bandwidth Per Hit	19.07KB
Average Bandwidth Per Visitor	157,15 KB

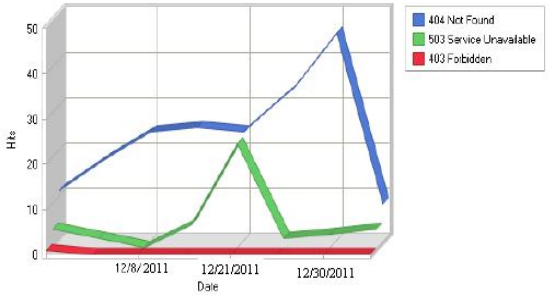


Fig2: Daily errors types

Table 2: Types of errors

Sr. No	Error	Hits
1	404 Not Found	189
2	503 Service Unavailable	43
3	403 Forbidden	1
	Total	233

Table 3: 404 Errors (Page Not Found)

Sr. No	Request/Referrer	Hits
1	/smartsyncpro/pad_file.xml No Referrer	38
2	/img/ssp_screenshot.html No Referrer	16
3	/smartsyncpro/history.html No Referrer	10
4	/smartsyncpro/registration.asp No Referrer	7
5	/t.php No Referrer	7
6	/smartsyncpro/screenshots.asp No Referrer	7
7	/)/ No Referrer	5
8	/smartsyncpro/screenshots.php No Referrer	5

Table 4: Other errors

Sr. No	Error	Request/referrer	Hits
1	503 Service Unavailable	/downloads/ssyncpro.exe http://www.smsync.com/ downloads	21
2	503 Service Unavailable	/downloads/ssyncpro.exe http://www.listsoft.ru/ programs/15371/	4
3	503 Service Unavailable	/favicon.ico No Referrer	4
4	403 Forbidden	/smartsyncpro/.html No Referrer	1

The following sections present results obtained when exploring the three research questions.

A. Classifier performance comparison

The two main variables affecting bug prediction performance that are explored in this paper are: (1) type of classifier (Naive Bayes), and (2) which metric (accuracy, F-measure) the classifier is optimized on. The four permutations of these variables are explored across all 11 projects in the data set. For SVMs, a linear kernel with standard values for slack is used. For each project, feature selection is performed, followed by computation of per-project accuracy, buggy precision, buggy recall, and buggy F-measure. Once all projects are complete, average values across all projects are computed. Results are reported in Table II.

B. Effect of feature selection

In the previous section, aggregate average performance of different classifiers and optimization combinations is compared across all projects. In actual practice, change classification would be trained and employed on a specific project. As a result, it is useful to understand the range of performance

TABLE II
AVERAGE CLASSIFIER PERFORMANCE ON CORPUS

Technique	Features Percentage	Accuracy	Buggy Precision	Buggy Recall	Buggy F-measure
Bayes F-measure	6.83	0.91	0.96	0.67	0.79
SVM F-measure	7.49	0.81	0.82	0.54	0.62
Bayes accuracy	6.92	0.86	0.92	0.53	0.65
SVM accuracy	7.83	0.86	0.96	0.51	0.65

TABLE III
NAÏVE BAYES USING F-MEASURE OPTIMIZED FEATURE SELECTION

Project Name	Features	Accuracy	Buggy Precision	Buggy Recall	Buggy F-measure	Buggy ROC
APACHE	465	0.92	1	0.56	0.72	0.78
COLUMBA	1618	0.9	0.99	0.67	0.8	0.83
ECLIPSE	802	0.98	0.98	0.79	0.88	0.88
GAIM	1065	0.86	0.96	0.65	0.78	0.83
GFORGE	2954	0.83	0.8	0.83	0.82	0.91
JCP	1041	0.95	1	0.77	0.87	0.89
JEDIT	847	0.9	0.99	0.73	0.84	0.88
MOZILLA	496	0.92	1	0.73	0.85	0.87
PLONE	277	0.91	0.98	0.57	0.72	0.84
PSQL	2504	0.87	0.88	0.54	0.67	0.78
SVN	438	0.94	0.96	0.56	0.71	0.78
Average	1137	0.91	0.96	0.67	0.79	0.84

achieved using change classification with a reduced feature set. Table III reports, for each project, overall prediction accuracy, buggy precision, recall, F-measure, and ROC area under curve (AUC) for the Na¨ıve Bayes classifier using Fmeasure optimized feature selection. Observing these two tables, a striking result is that three projects with the Na¨ıve Bayes classifier achieve a buggy precision of 1, indicating that all buggy predictions are correct (no buggy false positives). While the buggy recall figures (ranging from 0.54 to 0.83, with a average buggy recall of 0.69 for projects with a precision of 1) indicate that not all bugs are predicted, still, on average, more than half of all project bugs are successfully predicted. Figures 1 and 2 summarize the relative performance of the two classifiers and compare against the prior work by Kim etal. Examining these figures, it is clear that feature selection significantly improves both accuracy and buggy F-measure of bug prediction using change classification. As precision can often be increased at the cost of recall and vice-versa, we compared classifiers using buggy F-measure. Good Fmeasure's indicate overall result quality.

V. CONCLUSION

In this paper a new data mining model is proposed to predict the software bug estimation. The proposed model is implemented using open source technologies and applied over the open source MySql, Apache,Ecilipse bug repository. For the proposed work only two attribute of the bug summary and description are taken for similarity measurement based on which estimation prediction is done for the software bugs. Future scope for the related work could be analyzing the impact of other bug attributes for the software bug estimation and incorporating them for the prediction calculation to achieve more accurate results. Also semantic similarities between the software bugs can be measured and applied for meaningful bug estimation.

The obtained results of the study can be used by system administrator or web designer and can arrange their system by determining occurred system errors, corrupted and broken links. In this study, analysis of web server log files of smart sync software has done by using web log expert program. Other web sites can be used for similar kind of studies to increase their effectiveness. With the growth of web-based applications web usage and data mining to find access patterns is a growing area of research.

In the future, when software developers have advanced bug prediction technology integrated into their software development environment, the use of classifiers with feature selection will permit fast, precise, accurate bug predictions. With widespread use of integrated bug prediction, future software engineers can increase overall project quality in reduced time, by catching errors as they occur.

REFERENCES

[1] F. Masseglia, P. Poncelet, and M.Teisseire,"Using data mining techniques on Web access logs to dynamically improve Hypertext structure", 1999.
[2] Gabriek. Web usage mining and discovery of association rules from HTTP server logs.
[3] David A. Grossman, and Ophir Frieder, Information Retrieval: Algorithms and Heuristics (The Information Retrieval Series) (2nd Edition) (Paperback - Dec 20, 2004)
[4]http://www.w3.org/Daemon/User/Config/ Logging.htm#common-log file-format
[5] James Rubarth-Lay, "Optimizing Web Performance", 1996.
[6] Sunghun Kim, Kai Pan, E. James Whitehead, Jr., "Memories of Bug Fixes", SIGSOFT'06/FSE-14, November 5–11, 2006, Portland, Oregon, USA, 2006.
[7] Java, the open source object oriented programming API : http:// java.sun.com
[8] MySql, the open database management system : http:// www.mysql.com
[9] Xapian, the open source stemmer for text data cleaning: http://xapian.org/docs/stemming.html
[10] H. Zeng and D. Rine, Sept. 2004, Estimation of software defects fix effort using neural networks. In Proc. of the Annual International Computer Software And Applications Conference (COMPSAC '04), Hong Kong, IEEE.
[11] G. Canfora and L. Cerulo, April 2006, Supporting change request assignment in open source development. In Proc. of ACM Symposium on Applied Computing, pages 1767– 1772, Dijon, France.
[12] D. C˘ ubranic´ and G. C. Murphy, 2004, Automatic bug triage using text categorization. In Proc. International Conference on Software Engineering & Knowledge Engineering (SEKE), pages 92–97.