Managing Open Bug Repositories through Bug Report Prioritization Using SVMs

Jaweria Kanwal
Quaid-i-Azam University, Islamabad
kjaweria09@yahoo.com

Onaiza Maqbool
Quaid-i-Azam University, Islamabad
onaiza@qau.edu.pk

Abstract

Managing the incoming deluge of new bug reports received in bug repository of a large open source project is a challenging task. Handling these reports manually by developers, consume time and resources which results in delaying the resolution of crucial (important) bugs which need to be identified and resolved earlier to prevent major losses in a software project. In this paper, we present a machine learning approach to develop a bug priority recommender which automatically assigns an appropriate priority level to newly arrived bugs, so that they are resolved in order of importance and an important bug is not left untreated for a long time. Our approach is based on the classification technique, for which we use Support Vector Machines. Experimental evaluation of our recommender using precision and recall measures reveals the feasibility of our approach for automatic bug priority assignment.

1. Introduction

During the software development and maintenance process, errors (bugs) in a software system are reported by developers and users. In open source projects, usually an open bug repository is maintained to collect the bug reports from users and developers. Use of open bug repositories is particularly important in open source software (OSS) development environment, where the developers are distributed all over the world [14]. Bugzilla[1] is an example of an open bug repository which was introduced during the development of Mozilla[2] but now is widely used for many open source software systems

Open bug repositories are used for collecting the bug reports and also for allowing bugs to be identified and resolved at an appropriate time. These bug reports need to be analyzed carefully to determine, for example, whether the bugs are duplicate or unique, important or unimportant, and who will resolve them. This process is called bug triaging. The person who analyses the bug reports is called a triager. The task of triaging becomes very important for an open source project because the responsiveness of a project is usually measured by the number of outstanding bug reports in the repository and how quickly a bug report is addressed [2].

Managing the number of new bug reports received in a day in large open source projects is a challenging task because the number of reports is usually more than the available time and resources to triage them [1]. For example in Mozilla project, on an average 300 bug reports are received in a day. Thus Triagers can be overwhelmed by the number of reports that need to be triaged. Moreover, bug triaging is time consuming task because to triage a bug report, triager needs a lot of information about that bug [3]. If a triager starts reading all the reports one by one without prioritization of bug reports, it is possible that some important bugs are left untreated for a long time, negatively affecting the project.

Hence there is a need to prioritize the newly arrived bug reports by their order of importance [2], so that most important bugs are handled first and are allocated time and resources appropriately [8]. Although the bug report structure contains a field where the person who reports the bug may assign priority, but sometimes this field is left blank. Moreover, the reporter may not correctly assign the priority level as his opinion about the importance of a bug may be different from that of a triager, who has more information about the software as a whole.

In recent years, mining of software repositories such as source code repositories, bug repositories and email archives is on the increase. Mining techniques find hidden patterns from the data stored in these repositories and turn it into useful information and knowledge. Given the large number of new bug reports that a triager needs to handle, it would be useful to explore mining techniques to facilitate in triaging e.g. assigning priorities to newly arrived bug reports automatically. Machine learning classification techniques, which build models to categorize data into dif-

[1]www.bugzilla.org
[2]www.mozilla.com

ferent classes based on attributes, may be used to develop such a recommender by mining the bug data present in a bug repository.

In this paper, we propose the use of classification techniques to automate the process of assigning priorities to new bug reports by mining the bug data present in a bug repository. Support Vector Machines (SVMs) are used for classifying the new bugs according to priority by training the SVM using the bug data. Recommended priority can further be analyzed by the triager to confirm or refine the automatically assigned priority.

Thus our main contributions in this paper are:

- Proposing the machine learning based approach for automatic assignment of bug priority to new bug reports in open source bug repository.
- Exploring the bug report attributes that contribute more towards determining the priority of a bug.
- Evaluating the affect of training dataset size on the accuracy of bug priority recommender.

This paper is organized as follows. Section sec:RelatedReasearchWork presents the related research work for bug prioritization and automatic bug triage. Section 3 gives an overview of bug repository, bug life cycle and bug triage process. Section 4 presents our proposed classification framework. Section 5 describes the experimental setup. Section 6 details the results of our experiments. In Section 7, we conclude the paper with a discussion of results.

2. Related Reasearch Work

Bug finding tools find bugs from source code and prioritize them, but this prioritization is often faulty. Thus researchers have proposed ways to improve the prioritization. Sunghun et al. [12] analyzed the bug life time and the priority level assigned by a bug finding tool and reprioritized the bug categories (e.g. 'overflow' category) according to their life time. Life time of each bug was computed using software change history data. This work was further refined by reprioritizing the bug categories (found by bug finding tools) on the basis of bug category weight [13].

Kremenek et al. [11] also prioritized the bugs found by bug finding tools on the basis of frequency count of successful and failed checks. Tool's analysis decisions are used for classifying the checks into successful and failed checks. Z-ranking scheme is used for ranking the most important bugs first.

Anvik et al. [3] applied machine learning algorithms on bug report data to classify bug reports by developer name. Support vector machines, Nave Bayes algorithm and decision trees were used on Eclipse, Firefox and GCC bug report data, and SVM achieved high precision. In subsequent work, Anvik et al. [4] evaluated their approach by extracting the developer expertise from bug repositories and source repositories. Canfora et al. [5] also developed an approach to assign a developer to a new bug report, using historic information stored in bug and source code repositories.

Different machine learning approaches have been applied on bug repository data for automating bug triage [1, 4, 5], detection of duplicate bug reports [14], effort estimation for resolving a bug [15] and predicting the number of bugs for next versions [6, 16].

3. An overview of bug repositories

Most open source software projects have open bug repositories to collect bug reports from distributed users. Bug repositories are used to manage the bug reports so that they are assigned to appropriate developer or maintenance team [1]. As the developers and users of an open source project are distributed all over the world, a bug repository provides a forum for discussion to developers to communicate about the project development and enhancement and keep its users aware of the status of the reported bugs and enhancement features.

3.1. Bug report detail

Bug repositories collect detailed bug reports from users and developers. Structure of a bug report is more or less same across bug repositories; we describe the detail of Bugzilla as it is widely used by many OSS projects. A bug report contains a number of attributes some fields are categorical such as bug-id, date of submission, component (the software product and component in which bug appears), product, resolution, status, severity (how serious bug is), priority (how important bug is, represented normally as levels P1-P5 with P1 being most important), platform, operating system, reporter, assignee, cc-list, and some are text fields such as summary and long description. Some of the categorical fields are fixed at the time of report submission, e.g. bug id, report submission time and reporter name. Some fields such as product, component, severity, priority, version, platform and operating system are entered by the reporter but may be changed by the triager or developer if needed [3]. Other fields such as developer who resolves the bug, list of people who are interested in bug resolution, bug-status, and resolution, change throughout bug life time.

Bug-status and resolution fields track the life cycle of a bug. For example, when a bug report is placed in the repository its status is NEW. When a bug is fixed by developer its status is set to RESOLVED. After resolution, bug status is set to CLOSED or CONFIRMED (it is confirmed that appropriate resolution has been taken). Resolution type is recorded in the resolution field. For example, if a bug is resolved by the developer, resolution type is marked as

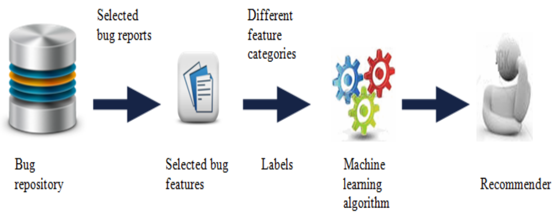

Figure 1. Proposed priority recommender

FIXED. If it is already fixed for another report it is marked as DUPLICATE. Text fields consist of summary and long description. Summary is the title of a bug report or short description of bug in one line written by reporter. In long description, problem is described in detail. User writes the problem he/she faced while using the software. Triager or developer also writes about the problem in detail after analyzing it.

3.2. Bug report triage

In a bug repository, bug triage is a process in which a triager takes a decision about the bugs entered in the bug repository by examining them in different ways. First of all, a triager makes sure that the reported bug is not a duplicate bug [14], and then it is checked for validity, i.e. is it a real bug or not. These two decisions are called repository-oriented decisions. The purpose of these decisions is to remove the bug reports that do not need to be resolved.

Remaining bug reports are analyzed for development oriented decisions. An important task that the triager performs is to examine severity and priority levels of the bugs, which are changed by him if inappropriate, so that important bugs will be given time and resources [10]. As pointed out in Section 1, assigning correct priority level is important to resolve more important bugs first. After this, triager writes comments for the bug and assigns this bug report to the appropriate developer to resolve the bug.

4. Proposed approach for priority recommender

In this section, we describe our classification-based approach for developing a priority recommender. Figure 1 illustrates our proposed priority recommender.

4.1. Classification-based recommender

Classification is the process of building a model by learning from a dataset that consists of training instances with associated class labels [7]. Each training instance consists of a set of features (attributes) that help in defining the characteristics of the class. A classification algorithm such as SVM finds the characteristics from the training instances for each class. This model is then used for predicting the class label of new instances.

Table 1. Sample bug report; features and class label

Features					Class
Bug-Id	Ver.	OS	Component	...	Priority
1	2.1	Windows XP	SWT	...	P3
2	2.2	Unix	Team		P1
3	3.1	Windows 2000	SWT		P4

Our approach is to build a classification model (bug priority recommender) from the information present in bug reports. Table 1 presents a sample bug report, features and class. Each bug report forms a training instance. The various fields within a bug report are its features. Priority represents the class label, with values P1-P5 forming the priority levels.

Existing bug reports are used to train the classification algorithm (recommender), which is then used for assigning a class label (priority level) to new bug reports. Bug reports that are marked as RESOLVED, CLOSED or CONFIRMED by the developers are selected for training because the priorities of such reports are set by the triager or developer after resolution of bug. The bug reports having status value NEW, UNCONFIRMED, ASSIGNED etc. are excluded from our training and testing dataset because priorities of these bug reports may not be authentic (until the triager analyzes the bug and developer fixes it, it is expected that priority may change). Features of a bug report available at report submission time are categorized into different feature categories. Different models can be built by varying the training features of the classifier.

4.2. Support Vector Machines

As the text data of a bug report is taken as training feature for classifier we selected SVM for classification. SVMs have shown promising results for text classification [9]. They have also been used for classification of software repositories in [1, 5].

Support vector machines build non-linear classification models from training data for each class. These models are then used for predicting the class of new instances [2]. SVMs transform the original data into higher dimensionality and find a separating hyperplane in the new mapped data. A hyperplane is found by using the support vectors and margins. If our data consists of two attributes then we can draw it in two-dimensional space as shown in Figure 2 [7]. This two dimensional data can be separated by various lines but SVM finds a line with maximum margin [7]. Similarly, for n dimensional data a separating n-1 dimensional hyperplane is found.

A separating hyperplane can be written as

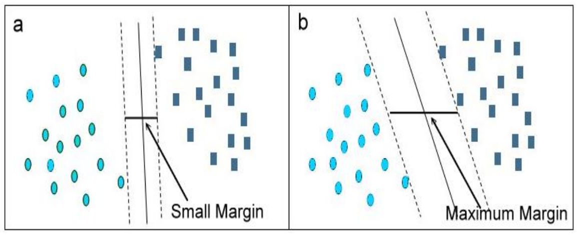

Figure 2. PSVM hyperplane with (a) Small margin and (b) large

$$w.x + b = 0 \tag{1}$$

where x represent the training instances (in our case bug report features) that lie on the hyperplane, w is a weight vector which determines the characteristics of the class, b is a scalar and "." represents the dot product. These are determined by the SVM from the training data set.

Classification of bug reports according to priority level is a single label multi-class problem. Internally SVM divides the bug reports into two classes so that one class is P1 and all other classes are treated as second class (bug reports having P2, P3, P4 and P5 priority levels). Bug reports of this class is again divided into two classes, one with bug reports of priority P2 and second class contains bug reports having P3, P4 and P5 priorities. Same process continues till the bug reports of all priority levels are separated.

5. Experimental setup

In this section we describe the test system used for our experiments, and the evaluation criteria.

5.1. Dataset and pre-processing

The Eclipse[3] project bug reports were used for our experiments. Eclipse is an open development platform with many users and developers. It was launched in 2001 by the IBM Corporation. Its projects are focused on building an open development Platform with platforms and tools for building and managing software through its lifetime. Eclipse bug reports we used for our experiments are from 2001 to 2006 for many products and components. Bug data is stored in two main versions: version 2 and version 3 and subversions. This dataset has been used for various experiments e.g. in [1] for automating bug triage, and in [15] for prediction of number of bugs. The Eclipse project data consists of 48 products. We used the Platform product for our experiments as this product contain a larger number of priority level-wise bug reports as compared to other products. Statistics of the dataset are presented in Table 2.

Originally the Eclipse bug reports were in the form of XML files. We developed an application in C# to extract our desired features. Extracted bug report features were product, component, version, platform, operating system, bug status, bug resolution, bug priority, bug severity, summary and long description.

[3]www.eclipse.org

Table 2. Statistics of each bug priority class in version 2 of Platform product of Eclipse

Bug	Priority class no. of instances
P1	705
P2	1073
P3	9441
P4	536
P5	327

Text attributes extracted from bug reports are bug summary and long description which contain a number of words that are not meaningful. So we applied a standard text categorization approach to transform the text data into meaningful representation. Stop and common words (e.g. of, the) were removed because these are unimportant and provide no information about the problem described in the bug report. Stemming is applied to transform a word into its ground form (e.g. testing and tests are reduced to test). Text of a bug report was converted into a feature vector which contains a word and its count.

5.2. Feature categorization

Bug report attributes can be divided in two main categories: categorical and text (summary and long description). Experiments were performed by taking different combinations of bug report attributes. We divided the bug report attributes that are available at report submission time into 5 categories as training features for classifiers. These are: CF: Categorical, SF: Summary, CSF: Categorical and Summary, TF: Summary and Long description, CSTF: Categorical, Summary and Long description.

Categorical attributes (CF) of a bug report that we used as training features for bug priority classification are component, platform, operating system and severity. Summary (SF) and long description (TF) attributes have been described briefly in Section 3.1. CSF is the combination of CF and SF categories and CSTF is the combination of CF and TF categories. Classifier is trained with different feature categories, to check which features contribute more towards bug priority classification. Our aim is to determine whether only categorical attributes (or only summary or text attributes) can train the classifier to achieve appropriate accuracy level for predicting bug priority of a new bug report or the features should be combined to achieve better results.

5.3. Training and test dataset

Platform product data is in two main versions. Classifier is trained on bug data of version 2. In our training dataset, number of samples for priority level P3 is very high as compared to other priority levels (See Table 2) which will over train the classifier for this class so we select equal number of samples for each bug priority class to train classifier. Maximum number of equal samples available in Platform product for each priority class is 327, making the total training dataset size to be 1635.

Version 3 bug data of Platform product is used to evaluate the classifier model (priority recommender). For version 3 bug reports we also have the actual class labels (priority levels assigned at bug resolution time), but to evaluate our recommender on real dataset we hid the class labels from the classifier and compare the actual priority levels with the predicted ones.

5.4. Evaluation criteria

We evaluate the classifiers' performance by using the precision and recall measures [7]. Precision of a class is the number of instances correctly classified as class 'A' divided by the total number of instances classified as class label A. Precision measures the percentage of correct predictions related to the predictions made by the classifier. Recall of a class 'A' is the number of instances correctly classified as class 'A' divided by the total number of instances in the dataset having class label 'A'. Recall measures the percentage of correct predictions related to actual classes. The precision of P1 class is 100% if a classifier predicts P1 class only for those instances which are actually P1 and it is 0 if classifier predicts P1 priority only for those instances which are not actually P1. Similarly recall of P1 is 100% if a classifier predicts class label P1 for all the instances which are actually P1 and recall is% if all the instances which are actually P1 are not assigned P1 class by the classifier regardless of whether other instances are assigned P1 class or not.

6. Experimental results and analysis

In our experiments, we varied the training features (as described in Section 5.2 and training data size for bug priority classification using SVM. The experimental results are detailed in this section.

6.1. Affect of training features

Precision and recall of different feature categories for each class are presented in Table 3, Table 4 and Figure 3, Figure 4. It can be seen that on an average, both precision and recall remain above 45%. It can also be seen that:

1. Precision and recall of P3 class are higher than other classes for all features categories.

2. On an average, precision and recall of CSTF are better than other feature categories.

Table 3. Precision of each bug priority class with different training features for Platform product

Features	P1	P2	P3	P4	P5	Average
CF	37	64	100	35	2	47.6
SF	20	32	92	31	6	36.2
CSF	30	53	100	37	9	45.8
TF	26	40	93	36	31	45.2
CSTF	33	58	100	47	32	54
Average	29.2	49.4	97	37.2	16	45.76

Table 4. Recall of each bug priority class with different training features for Platform product

Features	P1	P2	P3	P4	P5	Average
CF	23	28	96	86	2	47
SF	39	23	65	31	14	34.4
CSF	37	18	94	66	19	46.8
TF	57	33	72	30	43	47
CSTF	59	33	95	53	43	56.6
Average	43 27	84.4	53.2	24.2	46.36	

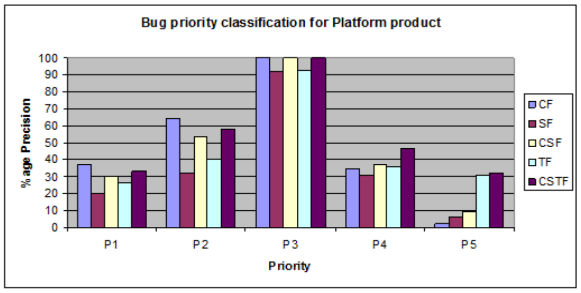

Figure 3. Precision of each priority class for different feature categories

Bug priority classifier should correctly classify the important bug reports so we need high precision and recall for P1 class.

1. Precision of P1 is less than 40% for all feature categories. This means that many unimportant reports are assigned P1 priority. In such a case, resolution of important bug reports may be delayed, since unimportant bugs may be deemed important.

2. Recall of P1 for TF and CSTF is more than 55% which shows that most of the important bugs are given P1 priority by these classifiers. Thus, relatively fewer important bugs are being categorized as unimportant, which is useful.

Figure 5 presents the average precision and recall for the five different feature categories. It can be observed that in TF category, precision and recall are 45% and 47% respectively but when TF is combined with the categorical fields, then precision and recall are increased by 9%. Similarly precision and recall of SF are increased by 10% and 12%

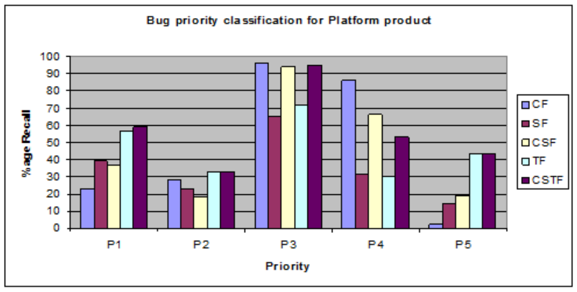

Figure 4. Recall of each priority class for different feature categories

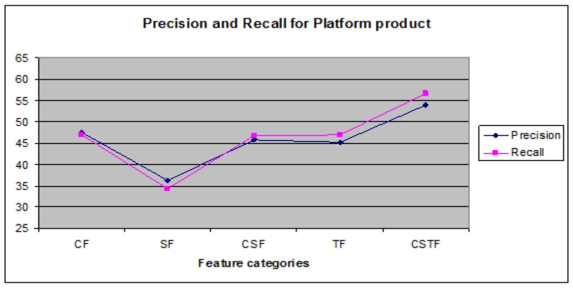

Figure 5. Average precision and recall for different feature categories

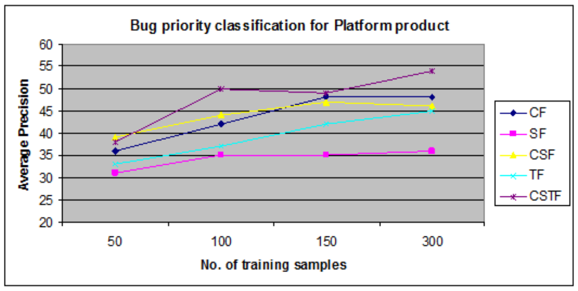

Figure 6. Average precision for varying data set size for different feature categories

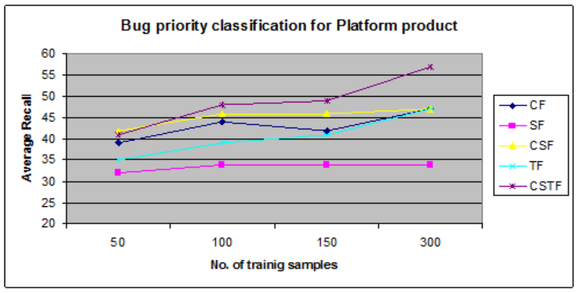

Figure 7. Average recall of bug priority classifier for varying data set size for different categories

respectively when combined with CF. This indicates that CF plays important role in improving the precision and recall.

It can be seen from Figure 5 that for SF category, precision and recall are lower than other feature categories. In summary field, type of the reported bug is described in few words which may have little information for priority wise classification of bug reports. That is why classifier performance for SF category is lower than other categories.

Figure 5 also shows that average precision and recall of CSTF category is better than for all other categories. This indicates that a combination of categorical and text attributes provides the most useful information, and when categorical attributes are combined with summary and text features, classifier performance is improved.

6.2. Affect of training data size

To evaluate the affect of training dataset size used in training the classifier, we trained classifier using 50, 100, 150 and 327 number of bug reports (training instances) for each class. Figure 6 and Figure 7 show the average precision and recall for various data sizes. It can be seen that:

1. For almost all feature categories, precision of bug priority classifier increases with increase in training data size, especially when size increases from 50 to 100 samples.

2. For size 50, average precision is lower than other sample sizes for all feature categories. For all feature categories, precision is highest for sample size 150 or 327.

3. Average recall increases with the increase in number of samples for all categories except CF category where recall at 150 is slightly lower than 100.

Average precision and recall are lower at sample size 50 than other sample sizes for all feature categories which shows that sample size 50 is not enough for training. For all feature categories average precision and recall is higher for either sample size 150 or 327 which indicates that sample size 150 may be enough for training bug priority classifier and a larger training sample size may not be needed.

7. Conclusion

This paper presented a classification-based approach to create a bug priority recommender, which assigns a priority level to new bug reports in a bug repository. Priority assignment assists triagers in resolving the important bugs first. The classification model is developed using SVMs, and bug report attributes are used as features for developing the model. Among different combinations of bug report features, precision and recall of CSTF category is better than all other categories. CSTF is a combination of categorical, summary and text features and thus contains relevant information useful for determining bug priority. The results indicate the feasibility of classification techniques for automatic priority assignment.

Classifier performance with varying training data size shows that precision and recall increase with the increase in data set size but at sample size 327, precision and recall remain almost same as for 150 samples, which indicates that increase in sample size after 150 samples has no significant effect on training.

In the future, we would like to extend this study to other test systems. We would also like to refine the evaluation criteria to obtain a better picture of the strengths and weaknesses of various feature categories.

References

[1] J. Anvik. Automating bug report assignment. In *Proceedings of the 28th International Conference on Software Engineering (ICSE), ACM*, pages 35–39, 2006.

[2] J. Anvik. *Assisting Bug Report Triage through Recommendation*. PhD thesis, University of British Columbia, 2007.

[3] J. Anvik, L. Hiew, and C. G. Murphy. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange, ACM*, pages 35–39, 2005.

[4] J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In *Proceedings Fourth International Workshop Mining Software Repositories ICSE Workshops MSR '07*, page 2, 2007.

[5] G. Canfora and L. Cerulo. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 1767–1772, 2006.

[6] T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering*, 31(10):897–910, 2005.

[7] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[8] I. Herraiz, D. M. German, J. M. Gonzalez-Barahona, and G. Robles. Towards a simplification of the bug report form in eclipse. In *Proceedings of the 2008 international Working Conference on Mining Software Repositories, ACM*, pages 145–148, 2008.

[9] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, 1998.

[10] T. Joseph, H L. Shan, X Chengdu, Spiros, Z, and Yuanyuan. Triage diagnosing production run failures at the users site. *ACM SIGOPS Operating Systems Review*, 41(6):144–157, 2007.

[11] T. Kremenek and D. Engler. Z-ranking: using statistical analysis to counter the impact of static analysis approximations. In *Proceedings of the 10th international conference on Static analysis*, SAS'03, pages 295–315, Berlin, Heidelberg, 2003. Springer-Verlag.

[12] K. Sunghun and D. Michael. Prioritizing warning categories by analyzing software history. In *Proc. Fourth International Workshop Mining Software Repositories ICSE Workshops MSR '07*, pages 27–30, 2007.

[13] K. Sunghun and D. Michael. Which warnings should i fix first? In *Proceedings of symposium on the foundations of software engineering, ACM.*, page 4554, 2007.

[14] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of 30th International Conference on Software Engineering*, 2008.

[15] C. Weib, R. Premraj, T. Zimmermann, and A. Zeller. Predicting effort to fix software bugs. In *Proceedings of the 9th Workshop Software Reengineering*, pages 1–2, 2007.

[16] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, page 915, 2007.