Mining Co-location Relationships among Bug Reports to Localize Fault-Prone Modules

Ing-Xiang CHEN(a), Chien-Hung LI(b), Nonmembers, and Cheng-Zen YANG(c), Member

SUMMARY  Automated bug localization is an important issue in software engineering. In the last few decades, various proactive and reactive localization approaches have been proposed to predict the fault-prone software modules. However, most proactive or reactive approaches need source code information or software complexity metrics to perform localization. In this paper, we propose a reactive approach which considers only bug report information and historical revision logs. In our approach, the co-location relationships among bug reports are explored to improve the prediction accuracy of a state-of-the-art learning method. Studies on three open source projects reveal that the proposed scheme can consistently improve the prediction accuracy in all three software projects by nearly 11.6% on average.

key words: bug localization, co-location shrinkage, fault-prone modules, statistical learning

1. Introduction

Effective debugging is a major issue in software quality assurance because it usually costs much time to find which software module contains bugs [1], [2]. Especially for a large-scale software project with overwhelming numbers of bug reports, many man-hours will be wasted [1]. To reduce such a huge amount of debugging cost, automated bug localization is in great demand [2]–[4].

In software maintenance tasks, bug localization is a procedure in which a software developer tries to find the bugs according to the bug reports. The major goal of automated bug localization is to locate the fault-prone software modules such that debugging can be facilitated. To identify fault-prone (FP) software locations, various studies have been conducted [2]–[14]. According to the initiation timing of prediction, these approaches can be divided into two categories: proactive and reactive. A prediction scheme is said proactive if it predicts FP software modules merely based on some static characteristics of software code itself. For example, program code complexity [5], code size [15], [16], object-oriented metrics [8], and revision history [11] are the metrics most commonly studied. The faulty patterns are also studied by Mizuno and Kikuno [13] to discriminate FP modules and non-fault-prone (NFP) modules using a spam filtering tool. With these proactive schemes, a program analyst can get a prediction list after each revision release to conduct related verifications for quality assurance. As shown in the past studies, the effectiveness of the proactive approaches depends on the validity of their fundamental hypothesis that complicated or frequently revised modules are usually fault-prone. However, many types of software faults are actually due to over-hasty typos or interface defects that are not related to code size [16]. In addition, many revisions are related to functional changes rather than bug fixing. Therefore, the effectiveness of the proactive approaches may be limited.

On the other hand, reactive approaches try to find bug locations based on the abundant information of historical bug reports. For each bug report, its descriptions are analyzed, and static software metrics including revision history and code complexity are mined to facilitate bug localization. However, computer-aided mining on bug reports is still in an early stage. The main difficulty is due to the challenges of building up comprehensive domain knowledge in bug report analysis. Therefore, many reactive approaches rely on advanced statistical learning techniques to extract FP characteristics of software modules, or code instrumentation frameworks to learn more code execution behaviors. For example, Jiang and Su use a CBI instrumentation framework and machine learning techniques to localize the bug locations [2]. Lukins et al. use a Latent Dirichlet Allocation (LDA) model to mine bug information from source code [4]. Although these approaches show their effectiveness on FP module prediction, they may not work well if part of the source code is unavailable. This absence situation may commonly exist due to the consideration of software legality protection.

In this paper, we propose a novel bug localization scheme called co-location shrinkage (CLS) utilizing the co-location relationships among historical bug reports and historical fixed module information to find possible FP modules for newly incoming bug reports. In our approach, two bug reports are said to have a co-location relationship if they have the same fixed module. From the aspect of each fixed module, bug reports with the co-location relationship of the same module can form an FP-correlated cluster. A corresponding classifier can then be built for each FP-correlated cluster to predict latent FP modules for newly incoming bug reports. The prediction results are ranked from the discriminability of the FP-correlated classifier. To enhance the discriminability of the classifiers, we further employ a shrinkage approach [17] that emphasizes the common semantic information for each FP-correlated cluster by aggregating the
semantics in accurately predicting FP modules for each bug. Therefore, the prediction accuracy is enhanced with the aggregated FP-correlations.

Compared with other bug localization schemes, CLS has three advantages. First, it can be applied to software projects in which part of the source code is unavailable. Users can thus use it in various debugging environments. Second, CLS does not need additional code instrumentation support. Therefore, the developer does not need to insert testing code for getting program execution behaviors. Third, the semantic information and the FP-correlations of the historical bug reports are used such that CLS can be applied to different levels of source code elements, such as packages, classes, and functions, as if the BRMS supports.

To verify the effectiveness of the CLS approach, we have implemented it based on Support Vector Machines (SVM) [18] classifiers because SVM has shown its superior performance of text mining in previous studies [19]–[21]. Three open source projects, Subversion (SVN) [22], AspectJ [23], and ArgoUML [24], were used to study the prediction accuracy. In the experiments, the CLS approach showed the improvements on an average of about 11.6% as compared with the SVM classifiers using only co-location information.

The rest of this paper is organized as follows. Section 2 reviews the related literature on FP module prediction and localization. Section 3 introduces the preliminaries for the essentials of BRMS and the problem statement. Section 4 presents the proposed co-location shrinkage approach. Section 5 describes the details of data collection and the experimental setup. Section 6 presents the evaluation results and discusses the threats to validity. Finally, Sect. 7 concludes the paper.

2. Related Work

In the last few decades, numerous proactive FP module prediction schemes have been proposed. Using different software metrics, these proactive schemes predict the latent FP modules without incoming bug information. These common metrics include code size [15], [16], defect densities [16], and object-oriented complexity measures [8], such as Weighted Methods per Class [25], [26] and Depth of the Inheritance Tree [25]. Historical software revision information is also studied because it reveals many clues for defect prediction [7], [9], [11]. Recent research further employs advanced information retrieval (IR) and machine learning (ML) techniques to identify FP modules. For example, Menzies et al. studied several classification techniques [27] and Mizuno and Kikuno adopt a spam e-mail filtering package to classify FP and NFP modules [13]. Although these proactive approaches have shown their effectiveness, most of them require the source code to be analyzed. In addition, many proactive approaches only consider the characteristics of software code itself rather than the semantic information of bug reports. Therefore, proactive approaches may have limitations in accurately predicting FP modules for each bug report.

Recently, reactive approaches have been proposed using advanced ML techniques and instrumentation frameworks to mine FP information. In [3], a statistical model called SOBER localizes FP modules by considering both correct and incorrect executions to identify bug-relevant patterns. Based on these patterns, SOBER uses a probabilistic ranking model to generate a rank list. In [2], a Co-operative Bug Isolation (CBI) instrumentation framework and ML techniques are used to localize FP locations. The inspected program code is first instrumented by CBI to collect execution profiles. Then an SVM model and Random Forests are used to learn the execution profiles to identify FP locations. On the other hand, Lukins et al. use a Latent Dirichlet Allocation (LDA) model to learn the probabilistic features of the inspected source code [4]. Then the bug report data are used as queries to predict the bug locations. Although these approaches have shown the effectiveness in bug localization, they require software source code to gather execution profile information or code distribution features. To the best of our knowledge, only the work in [12] proposes a PageRank-based social network model to predict FP modules without considering source code. However, this social network model cannot get consistent improvements over an SVM scheme in different software projects because there may be limited social network relationships between some bug reports and the fixed locations.

3. Preliminaries

A bug report generally consists of required metadata, a short summary, textual descriptions or error messages. Besides the submitted contents, it may have appended comments or responses from reporters. These textual descriptions usually provide valuable information for bug localization. Figure 1 depicts a bug report example from the ArgoUML project [24].

In a bug report management system (BRMS), a bug report is usually supervised and maintained in different phases before closed [28]. In the processing flow, it will be given different status of resolution in its life-cycle. In this study,
the resolution status “FIXED” is used as a clue to find the explicit links between bug reports and their corresponding fixed locations.

Figure 2 shows an example of relationships among bug reports and fixed locations in the Subversion project [22]. The relationships are defined as 1-to-many mappings. In this example, BR #618 is fixed on files wc, entries, and svn_wc, and BR #764 is fixed on files entries and status. The file entries apparently co-relates to both BR #618 and BR #764, and we say that both bug reports have the co-location relationship. This figure also shows that a newly incoming bug report BR may have several faulty locations $L = \{L_1, L_2, \ldots, L_n\}$ to be fixed. Thus, the prediction model of $L_x$ can be denoted as $P(BR_i \mid L_x)$ to describe the possibility of fixing $L_x$ for BR.

From our observations, we found that if BR is semantically related to BR #618 or BR #764, it tends to have the co-location relationships with them. That is, BR may be resolved by fixing the same file entries. Therefore, the mined co-location information can help localize the FP modules by providing the developers with a recommendation list of possible FP modules for each newly incoming bug report. For the simplicity, the granularity of FP modules is discussed on the level of files in this study. However, our approach can be applied to other levels of source elements such as packages, classes, and functions as if the BRMS supports.

4. Co-location Shrinkage

In this section, the details of the co-location shrinkage (CLS) scheme are described.

4.1 Classification Models

In this study, we consider SVM as the classifiers for its superior performance in many text mining applications [19]–[21]. To learn the co-location relationships, SVM classifiers are trained to learn the textual information of historical bug reports (HBR) and the fixed module information (L) in a high-dimensional information space. The SVM classifiers can then discriminate the testing data with the hyperplane model. Through finding the maximum margin hyperplane, SVM can reach greatest separation between the positive and negative classes.

In this research, a binary SVM is adopted to learn the co-location relationships. To get a ranking list of FP modules for each incoming bug report BR, the correct associations of HBR and L pairs are learned as the positive examples. The negative examples of the same number are randomly selected from incorrect pairs of bug reports and locations. Thereafter, the trained SVM classifier processes all possible (BR, L) pairs to get a ranking list according to their discriminative degree, where $L_j$ is the $j$-th software module.

To demonstrate the discrimination process, an SVM classifier with linear kernel functions, $f : X \in R^n \rightarrow R$, is used in Fig. 3 that shows a trained hyperplane testing each $(BR_i, L_j)$ pair as a positive one, $f(x) \geq +1$, or a negative one, $f(x) \leq -1$. The linear function is in the form of $f(x) = (w, x) + b = \sum_{i=1}^{n} w_i x_i + b$ where $(w, b) \in R^n \times R$. The linear SVM is trained to find the optimal values of $w$ and $b$ such that $\|w\|$ is minimized. Figure 3 also illustrates the hyperplane built with linear SVM classifiers that can identify the potential FP modules for an incoming bug report BR.

To support more accurate prediction, four kernels have been studied: a linear kernel, a polynomial kernel, a sigmoid kernel, and a radial basis function (RBF). The non-linear RBF kernel of $k(x, x') = \exp(-\gamma ||x-x'||^2)$, for $\gamma > 0$, slightly outperformed other kernels in our preliminary test. Henceforth, RBF is adopted with SVMlight [29] in this study.

4.2 FP-Correlated Cluster Shrinkage

Our hypothesis is that the existing co-location relationships imply that the bug reports co-locating to the same FP module are semantically co-related. The bug reports with the same co-location relationship thus form an FP-correlated cluster. Therefore, if a new bug report has similar semantic information to some FP-correlated cluster, the corresponding FP module may need to be fixed for solving this bug report.

To enhance the classification discriminability of each FP-correlated cluster, we use a shrinkage approach to adjust the semantic features of each historical bug reports of the same FP-correlated cluster toward the cluster centroid. The
centroïd of the co-located bug reports represent the common feature space of each cluster. Therefore, the discriminability of SVM classifiers can be enhanced because the FP-correlated clusters are now more obviously separated in the SVM hyperspace.

Figure 4 shows the co-location relationships among the bug reports shown in Fig. 2. The figure also presents the idea of clustering and shrinking the co-located bug reports on the entries and status clusters. It shows that CLS can be applied to strengthen the semantic concept of co-located bug reports.

Figure 5 shows the CLS algorithm in which it first calculates the centroïd of each co-cited location as the common feature space. For each bug report \( b \in L \), the term weight gained by TF-IDF (Term Frequency-Inverse Document Frequency) \([30]\) in \( b \) is replaced as \( b' = c \) with the term weight derived from \( c \).

In advance to learn the knowledge from historical bug report pairs, an SVM classifier is adopted with co-location shrinkage (CLS). Pairs of bug reports and their fixed locations are then trained and tested by the CLS scheme. When a new bug report comes, all the potential locations are paired up with it and each pair is then tested by the SVM classifier. Therefore, each test pair obtains a discriminative score by calculating its distance to the separation hyperplane. Finally, all the potential locations are ranked in a recommendation list according to the discriminative scores.

4.3 Process of Bug Localization

Figure 6 depicts the prediction process on FP modules. Historical bug reports and the related fixed locations are first respectively collected from the BRMS and CVS/Subversion. The associations between the historical “FIXED” bug reports and the fixed modules are then determined from the version archives identified from the messages describing changes, such as “Fixed issue #4321”. Besides, keywords in bug reports like “fixed”, “defect”, and “bug”, or patterns like “# and a number” are also used to extract the links.

After the bug reports and the fixed modules are made as pairs, the co-located pairs are clustered according to the locations. The CLS scheme is then applied to enhance the semantic discriminability of each FP-correlated cluster. By learning the pairs in the same FP-correlated cluster, a binary SVM is trained to predict the possible FP modules for incoming bug reports.

When a new bug report comes, all the potential locations are paired up with it and each pair is then tested by the SVM classifier. Therefore, each test pair obtains a discriminative score by calculating its distance to the separation hyperplane. Finally, all the potential locations are ranked in a recommendation list according to the discriminative scores.

5. Experimental Environment

5.1 Data Collection and Processing

Table 1 summarizes the collected data sets. The bug reports of three open source projects, Subversion \([22]\), AspectJ \([23]\), and ArgoUML \([24]\), were collected from Bugzilla \([34]\) and Tigris \([35]\). Subversion is a software configuration management (SCM) tool developed in C; AspectJ is an aspect object programming (AOP) extension developed in Java; ArgoUML is a UML tool implemented in Java. Their version archives have been well integrated in the BRMS.

To verify the accuracy performance, the fixed associations between bug reports and their corresponding fixed modules need to be identified first. Since the pattern matching scheme described in Sect. 4.3 may not provide sufficient association information \([36]\), text similarity measures between the bug reports and the change logs were also applied to discover more embedded associations.
this reason, the bug reports were treated as *bag-of-words* and preprocessed with common IR/NLP (Information Retrieval/Natural Language Processing) procedures, such as tokenization, stemming, and stopword removal [37]. In the experiments, the words in the description field of the bug reports were transformed into feature vectors with TF-IDF weighting. Figure 7 shows a snippet of the transformed feature vectors. In each feature vector of a bug report and its fixed locations, the location name is regarded as a label feature assigned with a weight.

5.2 Experimental Setup

In the experiments, bug reports were chronologically divided into 10 folds to ensure that the historical data were used to predict possible FP modules for incoming bug reports. Therefore, the training and testing cases were run incrementally by each fold, where the reports of the first fold were used for training and the reports of the second fold were regarded for testing. Then the first and second folds were trained to predict the third fold, and so forth until the last fold was tested. Consequently, the first 9-fold data were incrementally trained as historical bug reports and data of each latter fold were tested as the newly incoming bug reports.

In the experiments, the baseline was SVM with only co-located (FP-correlated) cluster information (CL). The baseline was then enhanced with the co-location shrinkage approach (CLS). Both schemes were executed in the same environment to provide users with a recommendation list of potential FP modules for each incoming bug report. In this study, at most 50 candidates were considered because for a project of over nearly one thousand source files, such as AspectJ and ArgoUML, a top-50 recommendation list covers a range to help developers focus on the top 5% of potential locations and reduce the search efforts over the overall FP modules. As in [11], [12], [38], the prediction accuracy was assessed by regarding a correct prediction in the recommendation list as a *hit*. Therefore, the accuracy is measured as the percentage of correct hits as follows:

\[
\text{Accuracy} = \frac{\# \text{ of hits}}{\# \text{ of hits} + \# \text{ of misses}}
\]

6. Evaluation and Discussion

This section presents the prediction accuracy performance of CL and CLS. We also discuss their applicability to other software projects and threats to validity.

6.1 Experimental Results

As in [11], [12], [38], the accuracy rates of all folds are averaged to show the prediction power of each scheme. Figure 8(a) presents the performance distribution of the Subversion project in the top-\(n\) recommendations, 1 \(\leq n \leq 50\). In Fig. 8(a), CLS consistently outperforms CL in most prediction ranking and achieves the best prediction accuracy of 56.32% in the prediction of 50 locations. Compared with CL, CLS relatively improves the prediction accuracy by 7.01% when the number of locations increases to 50.

In the second experiment, CL and CLS were evaluated on AspectJ. Figure 8(b) shows that CLS achieves the best prediction accuracy of over 40%. CLS relatively outperforms CL up to 5.62% with the top-50 recommended locations. In the case of a medium-scale project, such as AspectJ, CLS still improves CL in most prediction ranking.

For ArgoUML, the prediction accuracy of CLS increases modestly to 38.39% in a prediction of 50 locations.
Figure 8 (c) presents that the prediction accuracy of CLS is increasingly boosted and relatively outperforms that of CL to over 22% with the top-50 recommended locations. In a large-scale software project with over one thousand source files, the simple CL classifier fails to effectively learn the defect pairs for prediction. In this experiment, CLS consistently and significantly improved CL in the prediction performance.

To further understand the accuracy performance of each fold for all software projects, Figure 9 and Fig. 10 depict the accuracy rates in three projects for the top-20 and top-50 candidate lists. For some folds of data, e.g. fold 3 in AspectJ, both schemes have similar poor performance because more than 40% of the bug location data in the testing fold are different with the training data such that the prediction performance is hindered. This situation suggests that the location coverage of the training data sets can be further investigated to get better performance. Regarding the improvements of CLS, these figures show that CLS has better or comparable performance in over 90% cases. Especially, CLS consistently has significant improvements in last few runs. However, CLS does not have significant performance benefit in first few folds due to the scant amount of bug report information.

6.2 Threats to Validity

Although the CLS scheme has been validated to be effective in predicting the potential location of faults in Subversion, AspectJ, and ArgoUML, there is still much room for advanced investigation on other software repositories and different types of projects. Hence, there exist some threats to validity. First, the systems examined in the experiment might not be representative enough since many closed-source projects may result in different types of defect localities. Second, the prediction accuracy is based on learning the meta-information and textual contents in bug reports, and thus the quality of bug reports will profoundly affect the prediction performance. For example, several recommendations have been discussed in [21] to improve the quality of bug reports. Nonetheless, such kind of improvement requires that the reporting process in BRMS should be redesigned to enforce these recommendation. The period length of collecting bug reports can be another threat to validity because if it is too short, the machine learning approach may not be so effective due to the scant amount of the bug report information. However, the period is variant by projects. The feasible period is the time to collect 40 to 100 bug reports in our experimental projects. Besides, the partially collected faults in these three projects may also be threats to affect the prediction performance since only about 40% to 60% of faults were automatically collected for experiments. Especially for a recommendation list of 10 locations, the quality of the collected bug reports significantly affects its accuracy. It will be of most help to rapidly and accurately localize the FP modules in top 10 locations. In the future, data semantics about bug reports needs to be explored and extracted with advanced IR/NLP techniques to identify semantically relevant fault locations.

7. Conclusions

Most of the empirical studies on bug localization are based on source code complexity metrics and historical revision information analysis. In practical environments, however, source code information may not be available and bug reports become the critical information source to find possible FP modules. In this paper, we address bug localization with
a mining mechanism to learn the co-location relationships among bug reports. Our study has indicated that mining bug reports with the CLS scheme can achieve about 38% to 56% accuracy on average and improves CL by over 11% on average. It can be concluded that the co-location relationships and the FP-correlated centroids can indeed facilitate bug localization. Although there is much room for further performance improvement, we believe that localizing FP modules by mining bug reports is a promising direction for future research.

Several lessons have been learned, and they also indicate the future research directions. First, more important information embedded in bug reports can be further mined to facilitate bug localization, such as the structural and semantic information in the text descriptions. Second, bug reports may be written by users of different cultural backgrounds for some large-scale software projects in practical cases. Handling such diverse information will be a challenge from the aspect of NLP. Third, some bug reports are crucial to reveal more significant bugs. Accordingly, identifying such kind of bug reports can be helpful to discover more critical bugs. Furthermore, for a software project in its early stage, the current SVM approach has accuracy limitations due to the scant amount of bug report information. This is actually a challenge in current machine learning technology. To leverage this rare information problem, other kinds of information, such as source code characteristics, may need to be considered. In our future plan, various previous schemes will be implemented to conduct a more comprehensive performance study.

Acknowledgments

The authors would like to express many thanks to the anonymous reviewers for their precious suggestions and the National Science Council of the Republic of China, Taiwan, for supporting this research under Contract No. NSC 98-2221-E-155-033.

References

Ing-Xiang Chen received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science and Engineering, Yuan Ze University, Taiwan, in 2001, 2003, and 2009, respectively. His primary research interests include information retrieval, Web mining, software engineering, and information visualization.

Chien-Hung Li received the B.S degree from the Department of Computer Science and Engineering, Tamkang University, Taiwan in 2006. He is currently an M.S student in the Department of Computer Science and Engineering, Yuan Ze University, Taiwan. His primary research interests include information retrieval, Web mining, and engineering.

Cheng-Zen Yang received the B.S. and M.S. degrees from Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, in 1988 and 1990, respectively. Then he received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University in 1996. Since 1998, he joined Yuan Ze University, where he is currently working as an assistant professor. His research interests include wireless networks, embedded systems, information retrieval, and Web mining. He is a member of the ACM and the IEEE Computer Society.

Ing-Xiang Chen  received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science and Engineering, Yuan Ze University, Taiwan, in 2001, 2003, and 2009, respectively. His primary research interests include information retrieval, Web mining, software engineering, and information visualization.

Chien-Hung Li received the B.S degree from the Department of Computer Science and Engineering, Tamkang University, Taiwan in 2006. He is currently an M.S student in the Department of Computer Science and Engineering, Yuan Ze University, Taiwan. His primary research interests include information retrieval, Web mining, and engineering.