International Journal of Engineering Research and Modern Education (IJERME)
ISSN (Online): 2455 - 4200
(www.rdmodernresearch.com) Volume I, Issue I, 2016

USING PSO ALGORITHM: REMOVAL OF BUG TRIAGE USING DATA REDUCTION TECHNIQUE

P. Saravanan* & K. Adlin Suji**

* PG Scholar, Department of Master of Computer Applications, Dhanalakshmi Srinivasan Engineering College, Perambalur, Tamilnadu

** Associate Professor, Department of Master of Computer Applications, Dhanalakshmi Srinivasan Engineering College, Perambalur, Tamilnadu

Abstract:

Software companies spend over 45 percent of cost in dealing with software bugs. An inevitable step of fixing bugs is bug triage, which aims to correctly assign a developer to a new bug. To decrease the time cost in manual work, text classification techniques are applied to conduct automatic bug triage. In this paper, we address the problem of data reduction for bug triage, i.e., how to reduce the scale and improve the quality of bug data. We combine instance selection with feature selection by using PSO to simultaneously reduce data scale on the bug dimension and the word dimension based on fitness value. The results show that our data reduction can effectively reduce the data scale and improve the accuracy of bug triage. Data reduction (Dimensionality reduction) technique is used to determine the remove unimportant attributes fitness function in the Bug triage and reduce the dimension of the Bug Triage based on input word and fitness function evolved. Our work provides an approach to leveraging techniques on data processing to form reduced and high-quality bug data in software development and maintenance.

Index Terms: PSO & Fitness

1. Introduction:

Bug triaging is the process of determining, first, if issues reported in the bug tracker describe meaningful new problems or enhancements, and second, if they do, assigning them to the appropriate developers and target release milestones for further handling. Bug triaging has a particularly visible role in open source communities, where it is not uncommon to find projects with hundreds if not thousands of open bug reports. It is as important, however, in commercial projects, especially because such projects can ill afford to ignore incoming issues. Bug triaging is most frequently associated with dedicated meetings, scheduled regularly to prepare for the next phases of development. Whether, these phases are coarse-grained series of major releases (release planning) or a fine-grained setoff sprints in a more Agile project. However, bug triaging also takes place outside of such meetings.

It is very common for bug reports to be reassigned, because an initial assignment may rely on an inaccurate assessment of the root cause of a bug, a wrong impression may have been had of whom the expert is to resolve an issue, or a certain developer might become overloaded and the work must be re-balanced among the available developers. In the Eclipse project, for instance, about 44% of reported bugs were reassigned at least once. Bug triaging, then, is an on-going activity. The de-facto tools used today in support of bug triaging are bug trackers, with some of the more popular tools being JIRA, Bugzilla, Fog Bugz, and Trac. These tools are typically positioned as supporting the entire life cycle of a bug report, from developers and users first reporting a problem. To keeping track of the stage at which the bug is in its resolution cycle, to eventually closing the bug and providing the developer and organization, should they be interested, with a variety of statistics and reports. As a result, bug trackers serve many uses beyond being the repository for all open bug

reports, including as the developer's to-do list, a user-facing portal, and a key communication hub for large open source projects. Naturally, bug triaging is one of the activities supported by bug trackers, as the assignment of a bug to a certain developer and certain milestone is part of the life cycle of a bug.

Unfortunately, evidence is emerging that this support is rather limited. Particularly in the open source community, where triaging bug reports is a highly visible process and a responsibility that is distributed across the community, concerns have been openly voiced. As an example, a former contributor to the Mozilla project posted on his blog an explanation for no longer contributing to the project, stating that "Triaging is broken period. He continues by explaining that the bug tracker they use, Bugzilla, is ill suited for handling the large influx of bug reports from the community as part of a new release process.

2. Related Works:

Web script crashes and malformed dynamically generated webpages are common errors, and they seriously impact the usability of Web applications. Current tools for webpage validation cannot handle the dynamically generated pages that are ubiquitous on today's Internet. We present a dynamic test generation technique for the domain of dynamic Web applications. The technique utilizes both combined concrete and symbolic execution and explicit-state model checking. The technique generates tests automatically, runs the tests capturing logical constraints on inputs, and minimizes the conditions on the inputs to failing tests so that the resulting bug reports are small and useful in finding and fixing the underlying faults. Our tool Apollo implements the technique for the PHP programming language. Apollo generates test inputs for a Web application, monitors the application for crashes, and validates that the output conforms to the HTML specification. This paper presents Apollo's algorithms and implementation, and an experimental evaluation that revealed 673 faults in six PHP Web applications.

3. Proposed Work:

In a proposed system, simultaneously reduce the scales of the bug dimension and the word dimension and to improve the accuracy of bug triage. We propose a combination approach to addressing the problem of data reduction. This can be viewed as an application of instance selection and feature selection in bug repositories. Data reduction for bug triage aims to build a small-scale and high-quality set of bug data by removing bug reports and words, which are redundant or non-informative. The reduced bug data contain fewer bug reports and fewer words than the original bug data and provide similar information over the original bug data using Dimensionality reduction. This paper extends our previously presented work on the generation of bug report summaries with a task-based study to evaluate the usefulness of bug report summaries in the context of a real software task: bug report duplicate detection using Dimensionality reduction. PSO algorithm is used to calculate fitness value and predict the Efficient Retrieval result based on the fitness value. It also clarifies the description of the approach and provides a more thorough review of related work.

Advantages:

- ✓ Bug dimension: Instance selection can remove uninformative bug reports, meanwhile, we can observe that the accuracy may be decreased by removing bug reports.
- ✓ Word dimension: By removing uninformative words, feature selection improves the accuracy of bug triage. This can recover the accuracy loss by instance selection.

- ✓ Data privacy has become increasingly important in the Cloud environment.
- ✓ Reduce Computational Cost improve Quality of Bug Report

Modules Description:

- ✓ Preprocessing in BUG Triage
- ✓ Developer Query in Processing in BUG Triage
- ✓ Data Reduction Techniques
- ✓ PSO Algorithm

Preprocessing in BUG Triage: The technique for selecting a reduced set of features for large-scale data sets. The reduced set is considered as the representative features of the original feature set. First category split up into two find out the missing terms and duplicate results. When we check the proper solution for the problem lot of results occurred during searching. BUG triage is used for preprocessing the unwanted duplicate results and missing results. So this will reduce the 50% of the duplicate results.

Developer Query in Processing in BUG Triage: The aim of bug triage is to assign developers for bug fixing. Once a developer is assigned to a new bug report, the developer can examine historically fixed bugs to form a solution to the current bug report. For example, historical bugs are checked to detect whether the new bug is the duplicate of an existing one moreover, existing solutions to bugs can be searched and applied to the new bug. A problem for reducing the bug data is to determine for the developer query in order of applying instance selection and feature selection, which is denoted as the prediction of reduction orders.

Data Reduction Techniques: Data reduction is to reduce the scale and to improve the quality of data in bug repositories. We combine existing techniques of instance selection and feature selection to remove certain bug reports and words using Dimensionality reduction technique. A problem for reducing the bug data is to determine the order of applying instance selection and feature selection, which is denoted as the prediction of reduction orders. Reducing the Data Scale of data sets to save the labor cost, time Quality of developers. The aim of bug triage is to assign developers for bug fixing. Accuracy is an important evaluation criterion for bug triage. In our work, data reduction explores and removes noisy or duplicate information in data sets.

PSO Algorithm: To avoid the time cost of manually checking both reduction orders, we consider predicting the reduction order for a new bug data set based on historical data sets. The PSO algorithm is used to determine the fitness function in the Bug triage and reduce the dimension of the Bug Triage based on input word and fitness function evolved. This PSO helps to reduce time consuming and improves Bug Report Quality.

4. Experimental Analysis and Results:

Implementation is the process of translating design specification in to source code. The primary goal of implementation is to write source code and internal implementation. So that conformance of code to its specification can be easily verified, So that debugging, testing and modification are eased. The source is developed with clarity, simplicity and elegance.

The coding is done in a modular fashion giving such importance even to the minute detail so, when hardware and storage procedures are changed or now data is added, rewriting of application programs is not necessary. To adapt or perfect use must determine new requirements, redesign generate code and test exiting software/hardware. Traditionally such task when they are applied to an existing program has been called maintenance. There are many approaches available in software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to

as dynamic testing. Static testing is often implicit, as proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules.

Static testing involves verification, whereas dynamic testing involves validation. Together they help improve software quality. Among the techniques for static analysis, mutation testing can be used to ensure the test-cases will detect errors which are introduced by mutating the source code. Implementation is the process of translating design specification in to source code. The primary goal of implementation is to write source code and internal implementation. So that conformance of code to its specification can be easily verified, So that debugging, testing and modification are eased. The source is developed with clarity, simplicity and elegance.

The coding is done in a modular fashion giving such importance even to the minute detail so, when hardware and storage procedures are changed or now data is added, rewriting of application programs is not necessary. To adapt or perfect use must determine new requirements, redesign generate code and test exiting software/hardware. Traditionally such task when they are applied to an existing program has been called maintenance.

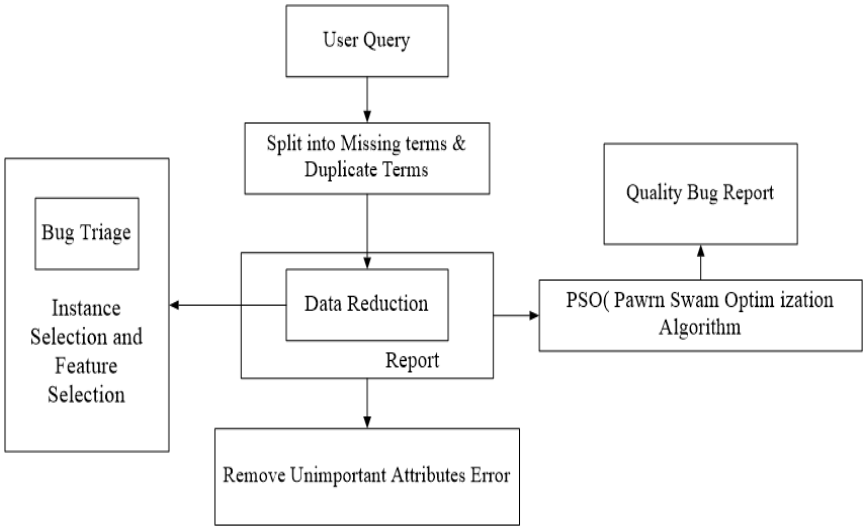
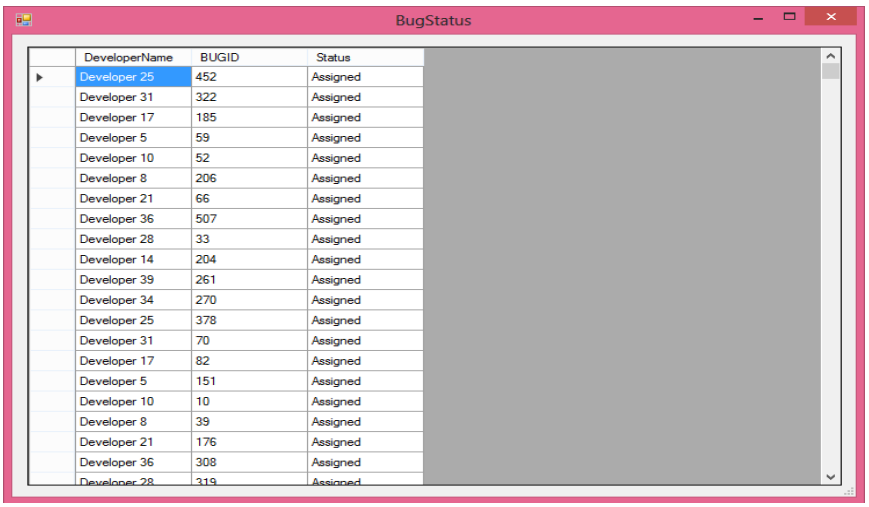


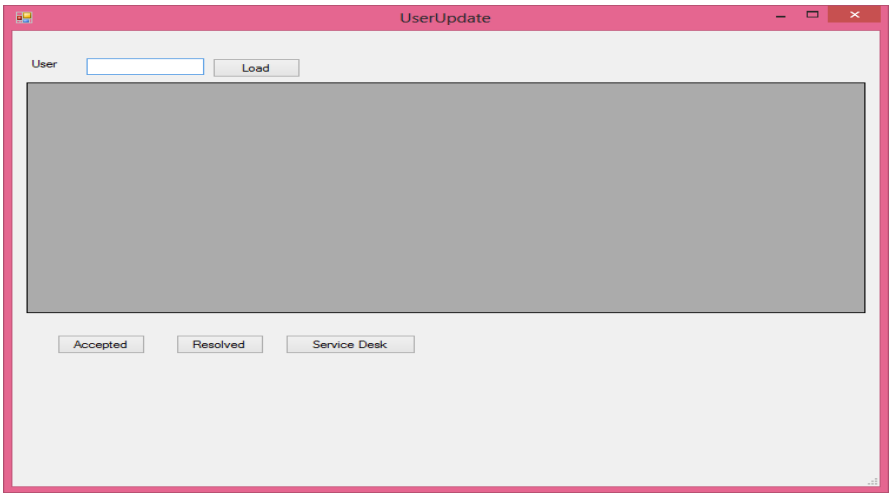
Fig.1. Detection of e-mail malware using bloom filter

Bug Status:



DeveloperName	BUGID	Status
Developer 25	452	Assigned
Developer 31	322	Assigned
Developer 17	185	Assigned
Developer 5	59	Assigned
Developer 10	52	Assigned
Developer 8	206	Assigned
Developer 21	66	Assigned
Developer 36	507	Assigned
Developer 28	33	Assigned
Developer 14	204	Assigned
Developer 39	261	Assigned
Developer 34	270	Assigned
Developer 25	378	Assigned
Developer 31	70	Assigned
Developer 17	82	Assigned
Developer 5	151	Assigned
Developer 10	10	Assigned
Developer 8	39	Assigned
Developer 21	176	Assigned
Developer 36	308	Assigned
Developer 28	319	Assigned

User Update:



5. Conclusion:

Bug triage is an expensive step of software maintenance in both labor cost and time cost. In this paper, we combine feature selection with instance selection to reduce the scale of bug data sets as well as improve the data quality. To determine the order of applying instance selection and feature selection for a new bug data set, we extract attributes of each bug data set and train a predictive model based on historical data sets. We empirically investigate the data reduction for bug triage in bug repositories of two large open source projects, namely Eclipse and Mozilla. Our work provides an approach to leveraging techniques on data processing to form reduced and high-quality bug data in software development and maintenance. In future work, we plan on improving the results of data reduction in bug triage to explore how to prepare a high quality bug data set and tackle a domain-specific software task. For predicting reduction orders, we plan to pay efforts to find out the potential relationship between the attributes of bug data sets and the reduction orders.

6. References:

1. J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.
2. S. Artzi, A. Kie_zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding bugs in web applications using dynamic test generation and explicit-state model checking," IEEE Softw., vol. 36, no. 4, pp. 474–494, Jul./Aug. 2010.
3. J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," ACM Trans. Soft. Eng. Methodol., vol. 20, no. 3, article 10, Aug. 2011.
4. C. C. Aggarwal and P. Zhao, "Towards graphical models for text processing," Knowl. Inform. Syst., vol. 36, no. 1, pp. 1–21, 2013.
5. Bugzilla, (2014). [Online]. Avaialble: http://bugzilla.org/
6. K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in Proc. 29th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval, Aug. 2006, pp. 43–50.
7. P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," IEEE Trans. Knowl. Data Eng., vol. 24, no. 6, pp. 1146–1150, Jun. 2012.
8. H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," Data Mining Knowl. Discovery, vol. 6, no. 2, pp. 153–172, Apr. 2002.

9. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in Proc. ACM Conf. Comput. Supported Cooperative Work, Feb. 2010, pp. 301–310.
10. V. Bolon-Canedo, N. Sanchez-Maro~no, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," Knowl. Inform. Syst., vol. 34, no. 3, pp. 483–519, 2013.
11. V. Cerveron and F. J. Ferri, "Another move toward the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule," IEEE Trans. Syst., Man, Cybern., Part B, Cybern., vol. 31, no. 3, pp. 408–413, Jun. 2001.