*Vol. 11, No. 4, December 2018, pp. 65 - 70* *ISSN 2006-1781*
*Shakirat Aderonke Salihu, Oluwakemi Christiana Abikoye, Amos Orenyi Bajeh and Abimbola Ganiyat Akintola (2018), A Survey of Empirical Studies on Performance Enhancement Features for IR-based Bug Localization Process*


https://afrjcict.net

# A Survey of Empirical Studies on Performance Enhancement Features for IR-Based Bug Localization Process

Shakirat Aderonke Salihu, Oluwakemi Christiana Abikoye, Amos Orenyi Bajeh and Abimbola Ganiyat Akintola
Department of Computer Science,
University of Ilorin,
Ilorin, Nigeria
*Email: salihusa1980@gmail.com,*
*abikoye.o@unilorin.edu.ng,*
*bajehamos@unilorin.edu.ng,*
*abimbola.akintola@gmail.com*

**ABSTRACT**
***Software is inevitable to have bugs. Localization of bugs has attracted many researchers due to its importance in software maintenance. Automation of Bug localization using Information Retrieval (IR) -Based approach has been proposed to attract more researchers due to its relatively low computational cost. Despite this automation, localization of bugs still takes the developers many hours or days to locate bugs. This paper tends to do a survey of some features that can be added to IR-Based bug localization process to enhance its performance and give a better result in terms of accuracy. The result from the six tools considered for this survey shows that there is an improvement in those with enhancement features compare with the baseline that has no enhancement features. The top N, MAP and MRR values of these tools outperform the technique without any enhancement.***

**Keywords:** *Bug Localization, Software Maintenance, Information Retrieval*

**African Journal of Computing & ICT Reference Format:**

Shakirat Aderonke Salihu, Oluwakemi Christiana Abikoye, Amos Orenyi Bajeh and Abimbola Ganiyat Akintola (2018), A Survey of Empirical Studies on Performance Enhancement Features for IR-based Bug Localization Process, *Afr. J. Comp. & ICT*, Vol.11, No.4, pp. 65 - 70.

## I. INTRODUCTION

It has always been an inevitable scenario for software to develop fault despite the energy and resources developers devote in producing software. Sometimes, the fault might not be detected immediately but as update continues, size and complexity grows and more use persists, there is tendency for the software to have bugs or faults. A software bug, or sometimes called fault, is a situation where software can no longer perform to its expectation. In other to fix bugs in software, developers set up a bug repository for collecting bug reports from users. A bug report is a detailed description, in a natural language text of the problems encountered in using the software. Bug reports are essential and vital for any software development. It is through the bug report that a customer is able to inform developers about the unexpected result encountered in using their software [1]. It usually contains a Bug Identifier (ID), Open date (the date a bug was submitted), fix date (the date a bug was fixed), summary and description (detailed information of a bug) [2].

In other to improve the reliability of systems, developers often allow users to submit bug reports to bug tracking system [3]. A Bug Tracking System (BTS) deals with the keeping track of reported bugs in software development. It is responsible for the management of bug repositories and help to document, assign, close and archive bug reports, which will later be sent to the developers of the software to locate and fix the bug. To manage bugs that appear in a software, developers often make use of a bug tracking system such as Bugzilla [4]. Open source development projects typically support an open bug repository to which both developers and users can report bugs [5].

Bug localization is an instance of concept location, where the change request is expressed as a bug report, and the end goal is to change the existing code to correct an undesired behavior of the software [6]. One of the most time-consuming tasks to resolve a bug report is to find the buggy files that are responsible for a reported bug. A system may contain thousands or more files and often only one or few of these files need to be changed to fix a bug [3]. The study carried out by [7] in which 374 bugs from Rhino, AspectJ and Lucene were analyzed found that 84-93% of the bugs resided in 1-2 source code files; this shows how difficult the task of bug localization can be. In cases of large software products, the number of bug reports in its bug repository may be so many that it will be very tedious for developers to resolve the large number of bug reports.

According to the study reported in [8], only Eclipse project received 4,414 bug reports in 2009 while [5] reported that every day, almost 300 bugs appear that need triaging and Mozilla projects received 51,154 bugs in 4 years. It is always challenging for developers to effectively and efficiently remove bugs, while not advertently introducing new ones at the same time [9]. Therefore, effective methods for locating bugs automatically from bug reports are highly desirable [10]. To overcome this issue, automated bug localization techniques, take as input bug reports and use textual information from the summary and description fields of these reports to find the buggy source code files [4].

The automation involves the use of Information Retrieval-(IR) based and Spectrum-Based approaches. The IR-based approach works by computing similarities between a reported bug and source code file [10] [11]. The source code files are then ranked based on their similarities to a reported bug. In spectrum-based approach, bugs are located via program execution information [12].

Despite these approaches for automatic bug localization, the developers still have a lot of source code files to search through before the buggy files can be found. In other to improve the performance of these approaches, there are some features identified which when combine with these two automated approaches will greatly increase the accuracy of the bug localization process. This paper reviewed some of these features and also shows the result of those that have used them before.

## II. LITERATURE REVIEW

This section discusses some identified features that developers normally take into consideration when carrying out bug localization manually. Some of these features are not incorporated into the automated process.

1. Stack Traces
2. Similar/Previously Fixed Bug Report
3. Version History
4. Components Structure
5. Dynamic execution Information

**Stack Traces:** Bug reports often contain stack trace information, which may provide direct clues for possible buggy files. Source files that are related to stack traces information in the bug report can be used to increase the ranking of the file based on the observation that the files covered by the stack trace are more likely to be bug-prone. Stack trace information refers to when an exception handling occur. Figure 1 presents an example of

a stack trace contained in a bug report of Eclipse, it contains a very long stack trace observed by the bug reporter. In the real fix of this bug, file *table.java* which also appears in the stack trace is actually modified [13]. This implies that buggy components can be easily located by analyzing the stack trace in bug reports.

**Bug ID 87855**
Summary NullPointerException in Table.callWindowProc
Here is a stack trace I found when trying to kill a running process by
pressing the kill button in the console view. I use 3.1M5a.
!ENTRY org.eclipse.ui 4 0 2005-03-12 14:26:25.58
!MESSAGE java.lang.NullPointerException
!STACK 0
java.lang.NullPointerException
at org.eclipse...Table.callWindowProc(**Table.java**:)
atorg.eclipse...Table.sendMouseDownEvent(**Table.java**:2084)
at org.eclipse...Table.WM LBUTTONDOWN(**Table.java**:3174)
atorg.eclipse...Control.windowProc(Control.java:3057)
atorg.eclipse...Display.windowProc(Display.java:3480)
...
atorg.eclipse.core.launcher.Main.run(Main.java:887)
atorg.eclipse.core.launcher.Main.main(Main.java:871)

Figure 1: Bug report with stack traces

**Previously Fixed bug**: This is often called similar report; it refers to the files responsible for a bug fixed recently which are more likely to be responsible for other bugs in the near future. That is, information on locations where past similar bugs were fixed could help locate the relevant files for the new bug [8]. Figure 2 shows an example of an older report with ID: 50303, which were reported nine months before the bug report with ID: 76138 shown in Figure 3. The two bug reports share common words which are highlighted and these can be a pointer to the buggy files.

**Version History:** This refers to historical data of changes made to source code files that are stored in a version control system during program evolution. This historical data can be used to improve bug localization performance [3] [14].

**Components Structure:** Bug reports and source codes files have structures. Bug reports have several fields including summary and description. Source code files can be split into class names, method names, variable names and comments. This structural information can be leveraged for bug localization [3] [15].

**Dynamic Execution Information:** It helps in complimenting the ranking of IR-based technique, it comprises of coverage, slicing and spectrum information. [16] stated that, spectrum information can help by ranking program entities based on suspiciousness scores and its ranking can complement ranking by IR-based techniques, also coverage and slicing can help to reduce the search space of IR-technique. Execution information can also help to boost the performance of IR, but it should be noted that coverage and slicing only help to reduce the search space but cannot rank the buggy files while spectrum suspiciousness score, if used with IR, will complement its ranking.

## III. COMPARATIVE RESULTS OF WORKS DONE ON PERFORMANCE ENHANCEMENT FEATURES

This section presents the result of other work done using some of the enhancements features discussed in section II and the improvement it has on bug localization process.

Discussion

**BLIA:** Is an IR-based bug localization process that uses Stack traces, similar bug report and code change history as enhancement features to boost the process. Their results show that enhancement features can be used to increase the performance of IR-based bug localization.
At the Top N rank (one of the performance metric for IR) which is set to 1, 5, and 10. BLIA outperform the baseline, where no enhancement features were added. The Mean Average Precision (MAP), which is used to rank all buggy files; and Mean Reciprocal Rank (MRR) which is used for finding the first buggy files. From the table, BLIA outperforms the MAP and MRR values of the baseline.

**BugLocator:** It uses previously fixed bug report as enhancement for the IR used. It also outperforms the values of the base line both at Top N rank, MAP and MRR across all projects considered.

**BRTracer:** This is based on the use of stack traces to boost the performance of IR. The results shows that stack trace, as an enhancement feature for bug localization process, can improve performance.

**Amalgam:** Version history, similar bug report and components structure were used as the enhancement in this approach. Both their top N rank, MAP and MRR results show the benefit of adding these features for IR-based bug localization process.

**Amalgam+:** The result of this approach which uses version history, similar bug report, stack traces, reporter

information and components structure as enhancement features.

**BLUiR:** Components structure of the source code files is used to enhance the performance of IR in this approach. Their result also outperforms the performance of baseline where no enhancement features was added.

The result of the six tools across all projects shows that the there is a wide gap between the baseline result without any enhancement and those with enhancement. For instance, in Table 1 Amalgam at top 1, 5 and 10 for AspectJ project has 44.4%, 65.4% and 73.1% while baseline has 12.59%, 23.78% and 28.67%. This percentage is considered low because the higher the percentage of the top N ranks, the better the result. Also, the MAP and MRR result for this approach has 0.33% and 0.54% while the baseline has 0.18% and 0.08%.

## IV. CONCLUSION

This paper identifies some enhancement features that can be used to boost the performance of IR-based bug localization technique. Some of the research works considered were able to get a better result when the features are added than using only IR-technique approach to bug localization.

It is therefore recommended that these features can be of utmost importance and also boost the performance of IR if added to the process.

## REFERENCES

[1] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Extracting structural information from bug reports," *Proc. 2008 Int. Work. Min. Softw. Repos. - MSR '08*, p. 27, 2008.

[2] Z. Shi, J. Keung, K. E. Bennin, and X. Zhang, "Comparing learning to rank techniques in hybrid bug localization," *Appl. Soft Comput. J.*, vol. 62, pp. 636–648, 2018.

[3] S. Wang and D. Lo, "Version history, similar report, and structure: putting them together for improved bug localization," *Proc. 22nd Int. Conf. Progr. Compr. - ICPC 2014*, pp. 53–63, 2014.

[4] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo, "BugLocalizer: integrated tool support for bug localization," *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2014*, pp. 767–770, 2014.

[5] J. Anvik, L. Hiew, and G. C. Murphy, "Who Should Fix This Bug ?," . *Proc 28th. Int. Conf. Softw. Eng. ICSE '06*, pp. 361-370, 2006.

[6] L. Moreno, J. J. Treadway, A. Marcus, and W. Shen, "On the use of stack traces to improve text retrieval-based bug localization," *Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014*, pp. 151–160, 2014.

[7] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended Comaparative study of association measures for fault Localization," *Journal of Software: Evolution and Process*, 26,2 pp 172-219, 2014.

[8] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," *Proc. - Int. Conf. Softw. Eng.*, pp. 14–24, 2012.

[9] W.E. Wong and V. Debroy, "Software Fault Localization," *Technology*, pp. 1–6, 2009.

[10] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," *Proc. 28th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 345–355, 2013.

[11] Shivani Rao and Kak Avinash, "Retrieval from Software Libraries for Bug Localization : A Comparative Study of Generic and Composite Text Models," *Proc. 8th work. conf. on mining soft. repo.(MSR'11), ACM,Waikiki,Honolulu, Hawaii*. pp. 43–52, 2011.

[12] Rui Abreu, Peter Zoeteweij, "An Evaluation of Similarity Coefficients for Software Fault Localization.pdf," *12th Pacific Rim Int. Symp. Dependable Comput.*, 2006.

[13] C. P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei, "Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis," *Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014*, pp. 181–190, 2014.

[14] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Inf. Softw. Technol.*, vol. 82, pp. 177–192, 2017.

[15] S. Rahman, K. K. Ganguly, and K. Sakib, "An improved bug localization using structured information retrieval and version history," *2015 18th Int. Conf. Comput. Inf. Technol. ICCIT 2015*, pp. 190–195, 2016.

[16] T. Dao, L. Zhang, and N. Meng, "How Does Execution Information Help with Information-Retrieval Based Bug Localization?," *IEEE Int. Conf. Progr. Compr.*, pp. 241–250, 2017.

| | |
|---|---|
| Bug ID | 50303 |
| Open date | 2004-01-20 20:55 |
| Summary | **Ant Editor** outline ”Link with **Editor**” |
| Description | Similar to the Java **Editor** it would be a nice enhancement to have a ”Link with **Editor**” toggle button for the **Ant Editor** outline page |
| Fixed Files | Org.eclipse.ant.internal.ui.editor.**AntEditor**.java<br>7 other files |

Figure 2: Previously Fixed Bug Reports

***Bug ID*:** 76138
**Open Date:** 2004-10-12 21:53:00

**Summary: Ant editor** not following tab/space setting on shift right

**Description:**
This is from 3.1 M2. I have **Ant->Editor**-> Display tab width set to 2, insert spaces for tab when typing checked. I also have **Ant->Editor**->Formatter->Tab size set to 2, and use tab character instead of spaces _unchecked_.
Now when I open a build,xml and try to do some indentation, everything works fine according to the above settings, except when I highlight a block and press tab to indent it. It's the tab character instead of 2 spaces that's inserted in this case.

**Fixed Files:**
org.eclipse.ant.internal.ui.editor.**AntEditor.java**
org.eclipse.ant.internal.ui.editor.**AntEditorSourceViewerConfiguration**.java

Figure 3: An Eclipse Bug report with fixed files

Table 1: Summary of the automated tools and their performance measures.

| **Project** [8] | **Approach** [13] [14] | **Top1 (%)** | **Top5 (%)** | **Top10 (%)** | **MAP** | **MRR** |
|---|---|---|---|---|---|---|
| AspectJ | BLIA | 37.7 | 64.4 | 73.2 | 0.323 | 0.491 |
| | BugLocator | 30.8 | 51.1 | 59.4 | 0.22 | 0.41 |
| | BRTracer | 39.5 | 60.5 | 68.9 | 0.286 | 0.491 |
| | AmaLgam | 44.4 | 65.4 | 73.1 | 0.33 | 0.54 |
| | AmaLgam+ | 49.4 | 72.7 | 80.3 | 0.40 | 0.60 |
| | BLUiR | 33.9 | 52.4 | 61.5 | 0.25 | 0.43 |
| | **Baseline No Enhancement** | **12.59** | **23.78** | **28.67** | **0.18** | **0.08** |
| SWT | BLIA | 68.4 | 82.7 | 89.8 | 0.637 | 0.746 |
| | BugLocator | 39.8 | 67.4 | 81.6 | 0.45 | 0.53 |
| | BRTracer | 46.9 | 79.6 | 88.8 | 0.53 | 0.595 |
| | AmaLgam | 62.2 | 81.6 | 89.8 | 0.62 | 0.71 |
| | AmaLgam+ | 63.3 | 80.6 | 89.8 | 0.62 | 0.71 |
| | BLUiR | 56.1 | 76.5 | 87.8 | 0.58 | 0.66 |
| | **Baseline No Enhancement** | **11.22** | **32.65** | **45.92** | **0.23** | **0.20** |
| ZXing | BLIA | 50.0 | 60.0 | 80.0 | 0.506 | 0.574 |
| | BugLocator | 40.0 | 60.0 | 70.0 | 0.44 | 0.50 |
| | BRTracer | N/A | N/A | N/A | N/A | N/A |
| | AmaLgam | 40.0 | 65.0 | 70.0 | 0.41 | 0.51 |
| | AmaLgam+ | 40.0 | 65.0 | 70.0 | 0.41 | 0.51 |
| | BLUiR | 40.0 | 65.0 | 70.0 | 0.39 | 0.49 |
| | **Baseline No Enhancement** | **20.0** | **35.0** | **50.0** | **0.28** | **0.27** |
| Eclipse | BLIA | N/A | N/A | N/A | N/A | N/A |
| | BugLocator | 29.1 | 53.8 | 62.6 | 0.30 | 0.41 |
| | BRTracer | 32.6 | 55.9 | 65.2 | 0.33 | 0.43 |
| | AmaLgam | 34.5 | 57.7 | 67.0 | 0.35 | 0.45 |
| | AmaLgam+ | 35.7 | 60.3 | 69.1 | 0.36 | 0.47 |
| | BLUiR | 32.9 | 56.2 | 65.4 | 0.33 | 0.43 |
| | **Baseline No Enhancement** | **6.86** | **16.91** | **23.93** | **0.13** | **0.09** |