© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Published version: Shinpei Hayashi, Takashi Kobayashi, Tadahisa Kato, "Evaluation of Cross-Lingual Bug Localization: Two Industrial Cases,", in Proceedings of the 39th IEEE International Conference on Software Maintenance and Evolution, pp. 495–499, 2023, https://doi.org/10.1109/ICSME58846.2023.00063

Evaluation of Cross-Lingual Bug Localization: Two Industrial Cases

Shinpei Hayashi
School of Computing
Tokyo Institute of Technology
Tokyo 152–8550, Japan
hayashi@c.titech.ac.jp

Takashi Kobayashi
School of Computing
Tokyo Institute of Technology
Tokyo 152–8550, Japan
tkobaya@c.titech.ac.jp

Tadahisa Kato
Research & Development Group
Center for Digital Services, Hitachi, Ltd.
Kanagawa 244–0817, Japan
tadahisa.kato.en@hitachi.com

Abstract*—This study reports the results of applying the cross-lingual bug localization approach proposed by Xia *et al.* to industrial software projects. To realize cross-lingual bug localization, we applied machine translation to non-English descriptions in the source code and bug reports, unifying them into English-based texts, to which an existing English-based bug localization technique was applied. In addition, a prototype tool based on BugLocator was implemented and applied to two Japanese industrial projects, which resulted in a slightly different performance from that of Xia *et al.

***Index Terms*—bug localization, cross-lingual information retrieval, machine translation**

I. Introduction

In software development, fixing bugs is time consuming. It is fundamentally difficult for developers to keep writing bug-free source code consistently, as bugs frequently occur. Bugs found by users or other developers are reported in bug reports and reviewed by developers so that they can be fixed [1]. When fixing a bug by referring to a bug report, the modules that need to be fixed must be identified to resolve the bug. This process, known as bug localization (BL), is a time-consuming and labor-intensive task in larger projects [2].

Automated BL techniques have been studied to improve the bug-fixing efficiency. There are several automated BL approaches, including dynamic analysis, static analysis, history analysis, and information retrieval (IR). Among these, all except for the IR-based approach require additional information, such as execution scenarios and a precise analysis of the program. The dynamic analysis approach includes fault localization techniques that record and analyze the test execution characteristics and test results for each case, thereby suggesting causes for the bugs. Unlike open-source software, users are not expected to perform an initial analysis on the conditions to reproduce defects or narrow down their causes. In many cases, a bug report is the only primary information on defects. In this study, the easiest IR approach that does not require additional information was used.

Most existing IR-based BL techniques assume that bug reports are written in English, whereas source files are written using English-based identifiers [3]–[5]. This is because identifiers in source code are considered a sequence of English terms such that the focus is on their lexical similarity to the English descriptions in bug reports. However, native non-English developers can also write bug reports in languages other than English, particularly for projects involving stakeholders from non-English-speaking countries. In such cases, the use of a non-English language is motivated to facilitate communication among developers. However, developers who receive non-English bug reports face difficulty in using existing BL approaches, which are designed for English text.

To mitigate this gap, Xia *et al.* proposed CrosLocator, a cross-lingual BL technique. It allows bug localization in reports written in languages other than English by applying machine translation. In particular, they examined the BL performance for Chinese bug reports [6]. Compared to the performance in the study of BugLocator evaluation [3], CrosLocator's performance was insufficient as a BL method after machine translation. Xia *et al.* concluded that cross-lingual IR is more challenging than normal IR, and cross-lingual BL is even more challenging than normal BL. However, to date, there have been few reported applications of cross-lingual BL. To the best of our knowledge, only the study by Xia *et al.* has been reported as applying the approach, and it focused solely on a Chinese open-source project.

This paper reports the results of a feasibility study on cross-lingual BL for two industrial projects in Japan. The main contributions of this study are as follows:

- Improving the performance of the cross-lingual BL approach by applying machine translation for source code, which includes Japanese texts, along with bug reports.
- Investigating the performance of two industrial projects.

The remainder of this paper is organized as follows: Section II explains IR-based BL approaches and the baseline cross-lingual BL approach briefly. Section III describes the extended cross-lingual BL approach used in this study. Section IV presents an evaluation of the approach using two industrial cases. Finally, Section V concludes the paper and outlines future research.

II. Preliminaries

A. IR-Based Bug Localization

In IR-based BL, source files and bug reports are considered documents, and their similarity is calculated using text analysis. IR-based BL calculates the similarity between a given

bug report and each source code, and outputs a ranked list of source files based on the calculated similarity.

1) VSM: The vector space model (VSM) is an IR approach that calculates the similarity between documents by vectorizing them. Term frequency-inverse document frequency (*tf-idf*) is typically used in vectorization, which considers the importance of terms in the documents.

Among various tf-idf computation methods, we adopted the following definition proposed by Zhou *et al.* [3]:

$$tf(t,d) = \log\left(\frac{c_{t,d}}{c_d} + 1\right),$$
$$idf(t) = \log\frac{|D|}{|\{\, d \mid c_{t,d} > 0 \,\}|}$$

where $d \in D$ denotes a document; $t \in T$ denotes a term; $c_{t,d}$ is the number of occurrences of the term t in document d; and c_d is the total number of terms in d. Term frequency tf increases as the number of occurrences of the term increases. The inverse document frequency idf decreases as the term appears in more documents. Assuming that terms with high generality are less important, the product $tf(t,d) \cdot idf(t)$ is used for the weight of t in d. Document d is vectorized as a $|T|$-dimensional vector $\vec{d}$ whose elements are the weights of each term.

The similarity score between a given source code d and a bug report q is calculated as the cosine similarity of the vectors:

$$\cos(d_1, d_2) = \frac{\vec{d_1} \cdot \vec{d_2}}{|\vec{d_1}||\vec{d_2}|},$$
$$\mathrm{VSMScore}(q,d) = \cos(q,d).$$

Large source files tend to have a high probability of containing bugs. However, in simple VSM, the score of a large source file tends to be lower than expected.

2) rVSM: The revised VSM (rVSM) [3] is a VSM that addresses the aforementioned issue by taking into account the number of terms in a file:

$$N(c) = \frac{c - c_{\min}}{c_{\max} - c_{\min}},$$
$$\mathrm{rVSMScore}(q,d) = \frac{1}{1 + e^{-N(c_d)}}\mathrm{VSMScore}(q,d)$$

where $N(c)$ is the linear normalization of the number of terms c to $[0,1]$; $c_{\max}$ and $c_{\min}$ are the maximum and minimum number of terms in the target document set, respectively. The larger the number of terms, *i.e.*, the larger the file size, the closer $N(c)$ is to 1, and the correction weight of VSMScore becomes larger. This means that $\mathrm{rVSMScore}(q,d)$ produces a higher value as the source code document d increases in size, thereby considering file size when calculating the similarity.

3) BugLocator: BugLocator [3], extended from rVSM, considers previous bug reports that have already been resolved.

$$\mathrm{SimiScore}(q,d) = \sum_{b \in B_d} \frac{\cos(q,b)}{n_b},$$
$$\mathrm{BugLocator}(q,d) = (1-\alpha)\, N(\mathrm{rVSMScore}(q,d)) + \alpha\, N(\mathrm{SimiScore}(q,d))$$

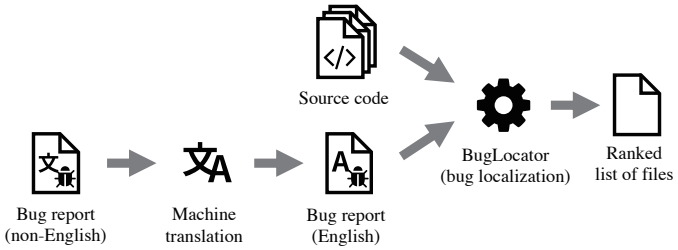

Fig. 1. Overview of cross-lingual bug localization via CrosLocator.

where B_d is the set of previous bug reports on source files that need to be fixed, and N_b is the number of fixed files required to resolve bug b. Parameter $\alpha \in [0,1]$ determines the ratio at which rVSMScore and SimiScore are combined. The case of $\alpha = 0$ is equivalent to rVSM and does not consider any previous bug information. When $\alpha = 1$, only the similarity with past bug reports is considered, and BugLocator increases when the target bug is similar to past bugs.

B. Cross-Lingual Bug Localization

Cross-lingual (or cross-language) IR retrieves documents that are written in a language different from the input query [7]. Several cross-lingual IR approaches have been proposed. Hayes *et al.* used Google Translate to apply machine translation to Italian sentences to obtain English sentences and recover software traceability [8]. Xu *et al.* performed corpus-based translation on a corpus that was pre-created by crawling websites and querying input texts, thereby using Chinese search queries on English websites to enable question retrieval [9]. Tonoike *et al.* extended term dictionaries by creating a corpus of compound nouns non-existent in dictionaries, thus improving the performance in Japanese–English translation estimation [10].

Xia *et al.* proposed CrosLocator, a cross-lingual BL technique applied to bug reports written in a non-English language by applying machine translation to the non-English bug report input. An overview of cross-lingual BL using CrosLocator is presented in Fig. 1. The input consists of source code based on English and a bug report written in a non-English language. First, machine translation is applied to the natural language descriptions in the bug report for translation into English. Next, BugLocator, an IR-based BL technique is run, with the source code and bug report translated into English as inputs, thereby obtaining a ranked list of source files that are likely to require fixing. BugLocator utilizes bug reports similar to the given bug report to find relevant source files based on the textual similarity between the bug reports and source files. Note that this figure only shows one machine translator, whereas the original CrosLocator uses multiple translators (see Section III for details).

III. Extension for CrosLocator

We extended CrosLocator to perform cross-lingual BL. We analyzed several software projects developed in Japan and found that comments and string literals in the source code

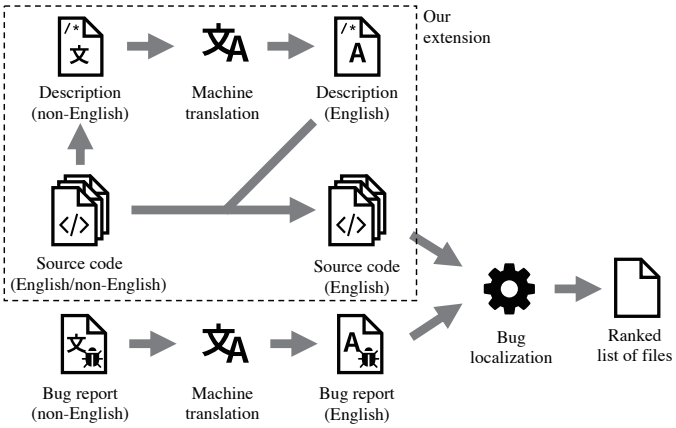

Fig. 2. Extension for CrosLocator.

sometimes included Japanese descriptions as well as bug reports. As Karampatsis *et al.* [11] reported, non-English terms can be found in strings and comments in source code due to the developer's primary language or internationalization purposes. Such Japanese terms in string literals or comments are processed directly as regular inputs by BugLocator in the later stage. In contrast, the Japanese descriptions in the given bug report were translated into English and input into BugLocator; therefore, even if both the bug report and source code contained similar Japanese descriptions, they did not contribute to the similarity degree.

Based on this observation, the similarity between each description was determined by unifying the descriptions into English in advance. An overview of the CrosLocator approach is shown in Fig. 2 with an extension for cases in which Japanese descriptions are also included in the source code. The source code was parsed to extract string literals and comment descriptions. Among the extracted descriptions, those containing at least one Japanese character were considered Japanese descriptions. Machine translation was then applied to translate them into English. Japanese characters were detected based on the range of character codepoints. The translated English descriptions were embedded in the same position from which the source code was extracted, replacing the original English descriptions. The entire source code was not translated directly because such an automated translation may break the identifiers and syntax of the source code. If a comment contained expressions corresponding to Japanese literals, only the literals were extracted and translated because translating the entire comment would lead to the same issue. Although this extension is simple, it can provide significant accuracy gains at low cost in cross-language BL when there are non-English terms in the source code.

A single machine translator was used for the CrosLocator approach. The original CrosLocator uses multiple localization results with multiple English texts obtained from multiple machine translators and computes the score by merging the multiple results through the data fusion technique [12]. However, sometimes it is not easy to have multiple translators available. Sending bug reports and source code to many external translation services may be impractical due to confidentiality concerns, especially in corporate use. Therefore, it was assumed that only a limited number of reliable machine translators are available for BL. For both confidentiality and practical use, only the cloud-based neural machine translation service contracted by the authors' organization was available for machine translation to be used in this study because the confidential software assets of the organization were to be shared.

TABLE I
TARGET PROJECTS

Projects	Main programming language	# source files	# bug reports
P_1	C#	929	8 / 186
P_2	Java	591	267 / 316

The BugLocator implementation bundled into Bench4BL [13] was used to implement the extended CrosLocator. We revised the original implementation, which only accepted Java programs as input source code, to allow inputs from other programming languages.

IV. EXPERIMENT

In this experiment, we aim to answer two research questions (RQs):

- ***RQ*$_1$ (Replication of CrosLocator approach):** Is the cross-lingual bug localization approach effective for industry projects?
- ***RQ*$_2$ (Effectiveness of the extension):** Does translating texts in source files contribute to the improvement of bug localization performance?

A. Experimental Setup

1) Dataset: The projects used in this experiment are presented in Table I. P_1 is a software product written mainly in C# and developed by a company. P_2 is written in Java and developed by the same company.

The bug reports maintained for each project were used in the study. Because an interview with a developer of the product in P_1 was conducted, as described below, the number of bug reports had to be limited to a small number. We filtered all 186 recorded bug reports based on the following criteria:

1) excluding bug reports unrelated to functional bugs (retaining 13 out of 186),
2) excluding bug reports for which bug fixes had not been completed at the latest code snapshot (retaining 9 out of 13), and
3) excluding a bug report in which a C# source file was not fixed when resolving the bug (retaining 8 out of 9),

thereby obtaining eight bug reports for P_1. In the case of P_2, 267 out of the 316 bug reports met the criterion of fixing at least one Java source file when resolving the bugs and were used for this study.

TABLE II
EXPERIMENTAL RESULTS

Project	Extension†	Base technique	MAP	MRR	Success$_{@5}$	Success$_{@10}$
P_1^d	Yes	BugLocator	0.224	0.322	0.375	0.500
	Yes	rVSM	0.158	0.248	0.375	0.375
	No	BugLocator	0.107	0.201	0.250	0.375
	No	rVSM	0.085	0.175	0.125	0.250
P_1^i	Yes	BugLocator	0.157	0.276	0.375	0.625
	Yes	rVSM	0.145	0.265	0.375	0.500
	No	BugLocator	0.112	0.139	0.250	0.375
	No	rVSM	0.097	0.178	0.125	0.250
P_2	(No)	BugLocator	0.174	0.270	0.390	0.461
	(No)	rVSM	0.145	0.220	0.326	0.397

† Extension in Section III: translating non-English texts in source files

These projects differ in the preparation of oracle lists of files related to the bug reports, *i.e.*, the ground truth files that should be identified by a BL technique from the given bug reports. In P_2, oracle files were automatically identified based on issue-commit linking. Bug reports and source code were managed using issue tracking and version control systems, respectively. The issue tracking system recorded in which commit the bug was resolved. The source files that were modified in the corresponding commit were identified as the files that needed to be fixed to resolve the bug. By associating these files with the bug report, the source files were determined as the ideal BL results corresponding to the bug report. However, this automated approach does not always identify the correct solution in a bug report. For example, incorrect or missing correspondences between bugs and fixing commits recorded in the issue tracking system or tangled commits [14] that include other changes along with the fixing of a bug can lead to an incorrectly deduced automatic solution. Additionally, in the BL process, it is practical to recommend files to developers that enhance their understanding of the bug, even if those files do not need to be fixed when resolving the bug. If only fixed files are considered correct, an evaluation from this perspective cannot be performed.

However, in P_1, we identified a mapping between bug reports and solution files based on an interview with an actual developer of P_1. The following two file types were identified for the eight bug reports:

- O^d: files containing *direct* code fragments to be fixed when resolving the bug, and
- O^i: files *indirectly* related and contributing to bug identification, but not containing direct code fragments.

To determine if the cross-lingual BL approach can identify not only direct code fragments but also indirect ones, we prepared an experimental treatment that included indirectly related files as oracles (P_1^i), in addition to another treatment that targeted only files containing direct bug locations as oracles (P_1^d). In the case of P_1^i, $O = O^d \cup O^i$ was used for the oracles.

The effect of our extension to translate non-English texts into source files introduced in Section III was evaluated using only P_1 because the source code of P_2 does not include any Japanese descriptions.

2) Evaluation Criteria: Mean average precision (MAP) [15], mean reciprocal rank (MRR) [16], and Success$_{@N}$ (percentage of rankings with at least one oracle file within the top N) were used as metrics to evaluate BL and other ranking-based tasks. These metrics were measured using trec_eval [17]. We used $N = 5$ and $N = 10$ for Success$_{@N}$, based on evidence of practitioners' expectations of BL [18].

B. RQ$_1$: Is the cross-lingual bug localization approach effective for industry projects?

The extended CrosLocator was applied to P_1^d, P_1^i, and P_2. The results are presented in Table II.

Overview. When BugLocator was used as the base bug localizer, Success$_{@5}$ for P_1^d was 0.375, indicating that at least one oracle file was recommended within the top five entries in 37.5% of bug reports. In addition, Success$_{@10}$ was 0.500, meaning that half of the results recommended at least one oracle file within the top 10 entries, highlighting the usefulness of the results. The MAP results for P_1^i were generally lower than those for P_1^d, indicating the difficulty in identifying more buggy files, including indirect oracles. The results using BugLocator for P_1^d produced better MRR scores than those for P_1^i, suggesting that BugLocator's feature of utilizing the relevant source files of similar bug reports may have had a negative effect when using indirect oracles.

Confirmation of the effectiveness of BugLocator compared to rVSM. Overall, the accuracy improved when BugLocator was used instead of rVSM. In other words, the performance improvement when using similar bug reports was effective as in BugLocator. The results for P_2 were similar to those of P_1.

Comparison between cross-lingual and non-cross-lingual settings. Xia *et al.* [6] reported that cross-lingual BL is challenging because the accuracy of the CrosLocator results (MAP of 0.116 for Ruby-China) was inferior to the accuracy reported in the paper proposing BugLocator for English bug reports (MAP of 0.22–0.545 for Zwing, SWT, AspectJ, and Eclipse) [3], although the target projects were different.

Although the values obtained in this experiment (MAP of 0.157 and 0.224 for P_1^d, P_1^i, and P_2) did not reach the ideal levels of BugLocator, they represented an improvement over the values reported for the extended CrosLocator. Considering

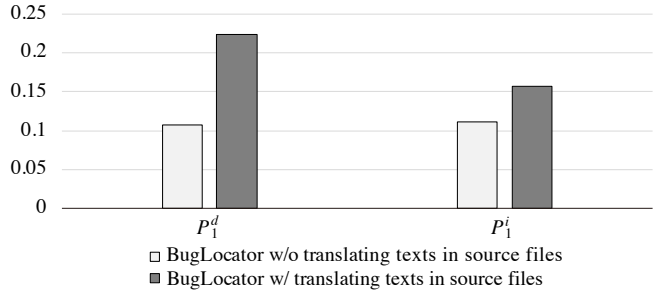

Fig. 3. MAP performance with and without text translation of source files.

the other evaluation metrics mentioned above, we believe that this approach is applicable to BL in Japanese.

> The cross-lingual BL with text translation of source files achieved reasonable performance.

C. RQ_2: Does translating texts in source files contribute to the improvement of bug localization performance?

The extended CrosLocator, with text translation of source files, was applied to the projects. The "Extension" column of Table II indicates whether translation was applied to the texts in the source files for the technique in Section III. Figure 3 provides a summary of the improvement in MAP performance when the text translation feature is available. When applying BugLocator, all evaluation metrics showed significant improvement when text translation was applied.

In a bug report of P_1, the ranking of the oracle file improved from 91st to 11th when text translation was applied. Japanese domain-specific terms corresponding to the terms in the bug report were included in the comments in the file, and their translation led to better lexical matching, resulting in an improvement in the ranking.

> The translation of texts in source files was effective in improving localization performance.

V. Conclusion

This study discusses the applicability of cross-lingual BL to Japanese descriptions. We applied the method to two industrial projects and achieved satisfactory results. To enhance the effectiveness of the BL approach, we plan to integrate a third-generation BL approach as the base bug localizer, thereby improving matching efficiency [19]. Moreover, we will extend the application of the cross-lingual BL approach to languages other than Chinese and Japanese. Lastly, we intend to apply this approach to additional projects, collecting cases that are based on the real localization settings employed by practitioners.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proc. OOPSLA Workshop on Eclipse Technology eXchange (eTX 2005)*, 2005, pp. 35–39.

[2] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," National Institute of Standards and Technology, Tech. Rep., 2002.

[3] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th International Conference on Software Engineering (ICSE 2012)*, 2012, pp. 14–24.

[4] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *Proc. 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)*, 2013, pp. 345–355.

[5] 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," in *Proc. 30th IEEE International Conference on Software Maintenance and Evolution (ICSME 2014)*, 2014, pp. 181–190.

[6] X. Xia, D. Lo, X. Wang, C. Zhang, and X. Wang, "Cross-language bug localization," in *Proc. 22nd International Conference on Program Comprehension (ICPC 2014)*, 2014, pp. 275–278.

[7] K. Kishida, "Technical issues of cross-language information retrieval: A review," *Information Processing & Management*, vol. 41, no. 3, pp. 433–455, 2005.

[8] J. H. Hayes, H. Sultanov, W.-K. Kong, and W. Li, "Software verification and validation research laboratory (SVVRL) of the University of Kentucky: traceability challenge 2011: language translation," in *Proc. 6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2011)*, 2011, pp. 50–53.

[9] B. Xu, Z. Xing, X. Xia, D. Lo, and S. Li, "Domain-specific cross-language relevant question retrieval," *Empirical Software Engineering*, vol. 23, no. 2, pp. 1084–1122, 2018.

[10] M. Tonoike, M. Kida, T. Takagi, Y. Sasaki, T. Utsuro, and S. Sato, "A comparative study on compositional translation estimation using a domain/topic-specific corpus collected from the web," in *Proc. 2nd International Workshop on Web as Corpus (WAC 2006)*, 2006, pp. 11–18.

[11] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code != big vocabulary: Open-vocabulary models for source code," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE 2020)*, 2020, pp. 1073–1085.

[12] S. Wu, *Data Fusion in Information Retrieval*. Springer Publishing Company, Incorporated, 2012.

[13] J. Lee, D. Kim, T. F. Bissyandé, W. Jung, and Y. Le Traon, "Bench4BL: Reproducibility study on the performance of IR-based bug localization," in *Proc. 27th ACM International Symposium on Software Testing and Analysis (ISSTA 2018)*, 2018, pp. 61–72.

[14] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Proc. 10th Working Conference on Mining Software Repositories (MSR 2013)*, 2013, pp. 121–130.

[15] G. V. Cormack and T. R. Lynam, "Statistical precision of information retrieval evaluation," in *Proc. 29th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006)*, 2006, pp. 533–540.

[16] N. Craswell, "Mean reciprocal rank," *Encyclopedia of Database Systems*, p. 1703, 2009.

[17] Text REtrieval Conference (TREC), "trec_eval," https://trec.nist.gov/trec_eval/, 2008.

[18] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*, 2016, p. 165–176.

[19] S. A. Akbar and A. C. Kak, "A large-scale comparative evaluation of IR-based tools for bug localization," in *Proceedings of the 17th International Conference on Mining Software Repositories (MSR 2020)*, 2020, pp. 21–31.