Bench4BL: Reproducibility Study on the Performance of IR-Based Bug Localization

Jaekwon Lee
SnT, University of Luxembourg
Luxembourg
jaekwon.lee@uni.lu

Dongsun Kim
SnT, University of Luxembourg
Luxembourg
dongsun.kim@uni.lu

Tegawendé F. Bissyandé
SnT, University of Luxembourg
Luxembourg
tegawende.bissyande@uni.lu

Woosung Jung
Seoul National University of Education
Seoul, South Korea
wsjung@snue.ac.kr

Yves Le Traon
SnT, University of Luxembourg
Luxembourg
Yves.LeTraon@uni.lu

ABSTRACT

In recent years, the use of Information Retrieval (IR) techniques to automate the localization of buggy files, given a bug report, has shown promising results. The abundance of approaches in the literature, however, contrasts with the reality of IR-based bug localization (IRBL) adoption by developers (or even by the research community to complement other research approaches). Presumably, this situation is due to the lack of comprehensive evaluations for state-of-the-art approaches which offer insights into the actual performance of the techniques.

We report on a comprehensive reproduction study of six state-of-the-art IRBL techniques. This study applies not only subjects used in existing studies (old subjects) but also 46 new subjects (61,431 Java files and 9,459 bug reports) to the IRBL techniques. In addition, the study compares two different version matching (between bug reports and source code files) strategies to highlight our observations related to performance deterioration. We also vary test file inclusion to investigate the effectiveness of IRBL techniques on test files, or its noise impact on performance. Finally, we assess potential performance gain if duplicate bug reports are leveraged.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Empirical software validation**; **Software evolution**; **Maintaining software**;

KEYWORDS

Reproducibility studies, bug localization, information retrieval

ISSTA'18, July 16–21, 2018, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5699-2/18/07...$15.00
https://doi.org/10.1145/3213846.3213856

ACM Reference Format:
Jaekwon Lee, Dongsun Kim, Tegawendé F. Bissyandé, Woosung Jung, and Yves Le Traon. 2018. Bench4BL: Reproducibility Study on the Performance of IR-Based Bug Localization. In *Proceedings of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18)*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3213846.3213856

1 INTRODUCTION

In software development and maintenance, debugging constitutes one of the most costly activities [5, 42, 50]. To alleviate debugging costs, the research community is striving to produce approaches and tools for automating various tasks. Among these tasks, bug localization [16, 25, 34, 35], i.e., the process of locating program elements which contain bugs leading to abnormal behavior, is a well-studied field. Bug localization leverages information such as issue reports [4] and stack traces describing a bug and its symptom, as well as possible reproduction steps, to identify which source code files, classes, functions, or code chunks are relevant to the bug.

The literature of bug localization proposes several approaches which leverage Information Retrieval (IR) techniques [7, 9, 15, 27, 28, 38] to identify potential bug locations by processing textual bug reports and source code files. Such approaches produce a ranked list of files where highly ranked files are supposed to be most likely to contain the reported bug. As Parnin and Orso pointed out, based on a developer survey, developers are strongly sensitive to the performance of such debugging tools: they do not find the tool useful when it does not help pinpoint the root cause of the bug early in the output ranked list [33].

State-of-the-art IR-based bug localization (IRBL) approaches are reported in the literature with increasingly high accuracy scores. Nevertheless, their benchmarking has not yet reached the level of maturity that is now required in several other sub-fields of software engineering. Our work contributes towards this maturity by (1) exploring a comprehensive evaluation of existing techniques with new large-scale datasets, (2) performing a reproducibility study on the performance of state-of-the-art techniques, and (3) investigating potential directions for improving IRBL performance (e.g., by matching the version of a bug report with source code files, assessing the impact of test file inclusion, or applying IRBL on duplicate bug reports).

Overall, this paper makes the following contributions:

- **A Reproducibility study with new subjects**: We examine potential overfitting scenarios by investigating the difference in performance when applied to the limited set of common (old) subjects used by most studies [37, 44–46, 52, 53] in the literature, and when applied to new subjects.
- **An Empirical Assessment of the impact of IRBL execution configurations**: Our study varies several different execution strategies of IRBL techniques to show the performance of the techniques in different evaluation environments such as version matching strategies and test file inclusion.
- **A Comprehensive benchmark for bug localization**: Finally, we package our datasets and detailed reproduction study results of state-of-the-art approaches into a new benchmark, Bench4BL[1].

Our findings include:

(1) The performance of IRBL techniques is actually higher than what is reported in the literature. We further show that experiments in the literature are also flawed since they were often performed on out-of-sync versions of the source code.
(2) While the differences in performance among the state-of-the-art techniques become less apparent with our large dataset, there is a high variability of performance across projects.
(3) In contrast to the common belief of the community [45], excluding test files is not necessary for the evaluation of IRBL techniques.
(4) Merging duplicate reports may achieve the performance gain.
(5) There is still room to improve the performance of IRBL.

The remainder of this paper is organized as follows. After explaining the background of IR-based bug localization and the motivation of this paper in Section 2, we describe the design of our study in Section 3. Section 4 illustrates, interprets the study results, and provides take-away messages for prospective users and researchers of IRBL. Section 5 discusses additional issues. After surveying the related work in Section 6, we conclude with directions for future research in Section 7.

2 BACKGROUND & MOTIVATION

Bug/Fault localization is a software maintenance activity in which a maintainer examines symptom information submitted by users, or obtained by running test cases, to locate suspicious resources (e.g., source code, configuration or data files) in a project. There are two leading research directions on bug (fault) localization, namely spectrum-based fault localization [1, 2, 10, 47] and IR-based bug localization [37, 44–46, 52, 53]. In the former, probable locations of faults are identified by computing ranking metrics, generally based on similarity coefficients and statistical techniques, on succeeding and failing test execution traces. In the latter, on the other hand, approaches only leverage source code information and bug report text to identify suspicious files using IR techniques such as Latent Dirichlet Allocation (e.g., [25]), Vector Space Model (e.g., [53]), Latent Semantic Analysis (e.g., [8]), Clustering (e.g., [24]).

IRBL techniques [37, 44, 46, 52, 53] are commonly known to exhibit low cost and to be scalable in practice since they require static information and, besides the bug report and the source code, they have no other external dependencies. The basic intuition for adopting IR technique for bug localization is that bug reports and project resources share common words. In other words, we can regard that a certain source file is suspicious if it contains words also written in a given bug report. According to the survey of feature location studies [13], IRBL is one of the tasks to locate feature locations; its type of analysis is 'textual', the input is a natural language query, the data source is source code, and the output is a subset of source code files in a project.

2.1 Performance Metrics

When evaluating IRBL techniques, many studies first collect ground truth data, which consist of bug reports already fixed by developers and the corresponding change set (i.e., files changed to fix a bug). Then, they apply the IRBL technique on this dataset and compare output results against the ground truth. The followings are common metrics used in most studies to assess an approach's performance:

- **Precision**: Also more accurately referred to as *Precision@k*, this metric represents an estimation of how many files are correctly recommended within given top k files. It is computed as follows:
$$P(k) = \frac{\text{\# of buggy files in top k}}{k} \quad (1)$$
- **Recall**: Also more accurately referred to as *Recall@k*, this metric estimates how many files are correctly recommended within given top k files over the actually fixed files by a developer for a given bug report. It is computed as follows:
$$R(k) = \frac{\text{\# of buggy files in top k}}{\text{\# of actually fixed files}} \quad (2)$$
- **Average Precision (AP)**: The average precision of a given bug report aggregates precision values of several positively recommended files as:
$$AP = \sum_{i=1}^{N} \frac{P(i) \cdot pos(i)}{\text{\# of positive instances}} \quad (3)$$
where N is the number of ranked files by an IRBL technique, i is a rank in the ranked list of recommended files. $pos(i)$ indicates whether the i-th file in the ranked list is a buggy file (i.e., $pos(i) \in \{0, 1\}$). For example, $AP = 0.5$, for a bug report with k actually fixed files, implies that an IRBL technique can make correct recommendations with 50% of probability within top k recommendations.
- **Mean Average Precision (MAP)**: The MAP is computed by taking the mean value of AP values across all bug reports:
$$MAP = \frac{1}{M} \sum_{j=1}^{M} AP(j) \quad (4)$$
where M is the number of all reports. $AP(j)$ is the average precision of report j. If MAP=p_{map}, at least one file is likely to be a correct recommendation for every $\frac{1}{p_{map}}$ files in the ranked list.
- **Mean Reciprocal Rank (MRR)**: This measure computes the mean value of the position of the first buggy file in the ranked list recommended by an IRBL technique, following this equation:
$$MRR = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{f\text{-rank}_i} \quad (5)$$
where M is the number of all bug reports and f-rank$_i$ means the position of the first buggy file in the ranked list for the i-th

[1] https://github.com/exatoa/Bench4BL

bug report. For example, assuming $MRR = 0.5$, an IRBL technique can locate at least one correct file to fix within top two recommendations (i.e., $\frac{1}{MRR} = \frac{1}{0.5} = 2$).

2.2 Motivation

We motivate this work as a contribution to the community for boosting the research on IR-based bug localization. We have indeed identified two key issues with the current state of research in IRBL, which justify a thorough reproducibility study for understanding which techniques indeed perform well and how the performance of such techniques can be realistically improved:

Performance of IRBL techniques is not solidly established: IRBL research has produced a number of state-of-the-art approaches in recent years. Unfortunately, performance assessments often focused on the same old and limited set of projects (such as old versions of JDT, AspectJ, etc.). Wen et al. [45] have also raised several threats to validity on the recorded performance: there could be some overfitting in the models due to some subject selection bias; and given the limited datasets, the data could be of poor quality to draw conclusions in the sense that they may not include interesting bug cases.

There are few perspectives on how to improve existing techniques: Most approaches in the literature focus on tuning the algorithms to gain fractions of precision performance points. In this work, we attempt to show that there is more room to demonstrate or improve the performance of techniques by consolidating the input data. For example, we investigate the consistency of project version with the bug report when applying IRBL techniques since bug reports indeed reflect the functioning of the program at a given time. We also examine whether including test files when identifying buggy files would help bug localization with IRBL techniques. In addition, duplicate bug reports may help us identify which features in bug reports are effective for IRBL techniques.

3 STUDY DESIGN

Our reproduction study explores different experimental scenarios to address the following research questions:

- **RQ1**: To what extent do IRBL techniques perform on up-to-date subjects?
- **RQ2**: What is the impact of version matching on the performance of IRBL techniques?
- **RQ3**: To what extent are IRBL techniques sensitive to the inclusion of test code files?
- **RQ4**: What potential performance gain can be reached by leveraging duplicate bug reports?

The objective of RQ1 is to (1) reproduce the results of IRBL techniques with the outdated subjects from the literature and (2) produce another set of results of the techniques with larger and recent subjects collected from active projects. With these two objectives, we can eventually establish whether some performance results recorded in the literature were actually over-fitted to a specific set of subjects.

Throughout RQ2 and RQ3, we investigate how IRBL techniques perform under different execution settings. Our study first (RQ2) compares two different version matching strategies: (1) assuming that an IRBL technique identifies files in a *single* version of the target program for all bug reports and (2) each bug report corresponds to a specific version of the program. Second (RQ3), we give two different search spaces to IRBL techniques: source code files with and without test files.

RQ4 aims at estimating the usefulness of duplicate reports for IRBL. Since duplicate reports contain different descriptions for a single bug, they could help improve performance.

To answer these research questions, we selected IRBL techniques whose implementations were readily available and usable (cf. Section 3.1). We then collected data from a large and diversified set of open source projects (cf. Sections 3.2 and 3.3); the dataset is publicly available at Bench4BL (https://github.com/exatoa/Bench4BL). In Sections 3.4 and 3.5, we define execution strategies for IRBL techniques to evaluate them in various configurations.

3.1 IRBL Techniques

We identified six recent state-of-the-art techniques that target Java projects:

- (2012) - **BugLocator** [53] leverages similar bug reports that have been previously fixed and relies on revised Vector Space Model (rVSM) for the recommendation.
- (2013) - **BLUiR** [37] extracts code entities such as classes, methods, and variable names from bug reports and leverages them to localize files.
- (2014) - **BRTracer** [46] analyzes stack traces shown in bug reports to improve bug localization accuracy.
- (2014) - **AmaLgam** [44] utilizes revision history in addition to similar reports and code entities.
- (2015) - **BLIA** [52] combines information such as similar reports, revision history, code entities, and stack trace information all together to improve the performance of IRBL.
- (2016) - **Locus** [45], the most recent technique, leverages code change information.

We select the above six techniques because they are frequently adopted to perform comparative studies against each other. Locus is one of the most recent techniques and its evaluation compares with BRTracer, BLUiR, and AmaLgam. The work of BLIA compares its performance with BugLocator, BLUiR, BRTracer, and AmaLgam. In the work of AmaLgam, the authors evaluated its performance comparing with BugLocator and BLUiR. The performance of BRTracer and BLUiR are compared with BugLocator in their work.

Default parameters: We use the default parameters in the literature by the approach authors for their assessment experiments. As recent IR-based studies [12, 32] pointed out, configuration and parameters of each IRBL technique might have a significant impact on the performance of bug localization. Since our goal is to reproduce the techniques and their results without any modification, we use the same configuration and parameters specified in each paper. If there are multiple configurations or parameters are available, we take the best one that was reported to outperform other settings. We provide more details in Section 6 for these techniques.

Specifically, we leverage the following configurations and parameters. For BugLocator, our setup includes the use of the revised Vector Space Model (rVSM), similar bug reports, the weighting factor α (for similar reports) of 0.3, and a logistic function as the length function. In case of BLUiR, we utilize TFIDF with term frequency parameter $k_1 = 1.0$ and document scaling factor $b = 0.3$, select the Krovetz stemmer, enable exact identifier name indexing and code structure modeling, and leverage similar bug reports. The execution setup for BRTracer leverages rVSM and similar bug reports. For AmaLgam, we use the same setup with BugLocator since the technique is based on BugLocator. This technique takes two additional parameters: version history length $k = 15$ and composer component ratio $b = 0.3$ by default. In case of BLIA, its configuration uses rVSM and three control parameters $\alpha = \beta = 0.2$, $k = 30$ by default as specified in [52]. For Locus, we use VSM and control parameter $\lambda = 0.5$ by default. Refer to the corresponding papers of the six techniques for more details of each parameter.

In addition, we run the above six techniques with the following environment: the machine that has an eight-core with 3.6GHz Intel processors and 16GB memory, and the 64-bit Ubuntu 16.04.

3.2 Subjects and Data Extraction

The first part of Table 1 lists 46 open source projects, denoted as *new subjects* collected for our study and which are (1) written in Java, (2) with publicly available bug reports, and (3) having at least 20 source code files in one of its version. We applied these criteria to the projects of Apache[2], Spring[3], and JBoss[4]; we selected those software communities since they are the largest ones with projects written by Java and well-managed together with issue tracking systems such as Jira[5].

In addition, we consider projects commonly used in the literature of IRBL [37, 44, 46, 52, 53] as listed in the second part of Table 1 as *old subjects*. Since the previous studies have used slightly different data sets from each other, we collect the union set of reports and source files as long as those data are available. Note that we have collected bug reports and source code files for Eclipse JDT and PDE instead of Eclipse itself since the repository of Eclipse is now separately managed for each sub-project [45]. Although additional benchmarks are available in [11, 14], we focus on existing subjects widely-used in IRBL studies.

In accordance with the settings of previous studies [37, 44–46, 52, 53], we collect only source code files (i.e., `*.java` files) from each subject. As shown in Table 1, 61,431 files are collected from 46 new subjects and 19,475 files from five old subjects. Note that the number of files is computed for the version with the maximum number of source files for each subject: the actual number of files could be different for each specific version (cf. Section 3.4 for multi-version collection details).

For each subject in Table 1, we collect bug reports from the corresponding issue tracking systems (ITS). Among all reports available in an ITS, we only select bug reports that are explicitly classified as "*Bug*" by developers and marked as **FIXED** with explicit file changes, yielding 9,459 reports from the 46 new subjects and 558 reports from the five old subjects. From each collected bug report, we extract information only available when it initially submitted, i.e., summary (or title), description, and reporter. We leave out extra information such as comments.

[2]Apache, http://www.apache.org/
[3]Spring, https://spring.io/
[4]JBoss, http://www.jboss.org/
[5]Jira, https://www.atlassian.com/jira

Table 1: Projects used in our study.

Group	Subject	# Source files (Max)	# Major versions	# Bug reports	# Duplicate reports
New subjects					
Apache	CAMEL	14,522	60	1,469	50
	HBASE	2,714	70	836	80
	HIVE	4,651	21	1,241	270
	CODEC	115	9	42	2
	COLLECTIONS	525	7	92	16
	COMPRESS	254	15	113	9
	CONFIGURATION	447	11	133	4
	CRYPTO	82	1	8	0
	CSV	29	3	14	0
	IO	227	13	91	7
	LANG	305	16	217	23
	MATH	1,617	15	245	8
	WEAVER	113	1	2	0
JBoss	ENTESB	252	3	47	1
	JBMETA	858	5	26	1
	ELY	68	3	25	1
	SWARM	727	6	58	1
	WFARQ	126	1	1	0
	WFCORE	3,598	16	361	8
	WFLY	8,990	11	984	27
	WFMP	80	1	3	0
Spring	AMQP	408	33	108	3
	ANDROID	305	2	11	0
	BATCH	1,732	33	432	3
	BATCHADM	243	4	20	0
	DATACMNS	604	33	158	15
	DATAGRAPH	848	4	60	2
	DATAJPA	330	38	147	11
	DATAMONGO	622	40	271	19
	DATAREDIS	551	17	49	1
	DATAREST	414	23	132	12
	LDAP	566	5	53	1
	MOBILE	64	3	11	0
	ROO	1,109	15	714	72
	SEC	1,618	42	541	68
	SECOAUTH	726	7	101	0
	SGF	695	19	107	1
	SHDP	1,102	9	45	0
	SHL	151	3	11	1
	SOCIAL	212	4	15	0
	SOCIALFB	253	5	15	0
	SOCIALLI	180	1	4	0
	SOCIALTW	153	5	8	0
	SPR	6,512	12	130	73
	SWF	808	20	134	5
	SWS	925	25	174	12
Sub-total		61,431	690	9,459	807
Old subjects					
Old subjects	AspectJ	6,485	1	286	35
	JDT	6,842	1	94	24
	PDE	5,273	1	60	20
	SWT	484	1	98	57
	ZXing	391	1	20	-
Sub-total		19,475	5	558	136
Total		80,906	695	10,017	943

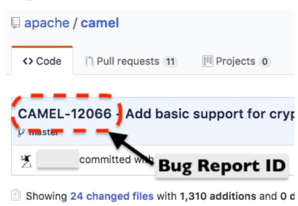
Duplicate bug reports: We searched the ITS for duplicate bug reports of all bug reports. Relationships between master bug reports (i.e., those marked as fixed) and duplicate bug reports are $1 \leftarrow n$ (i.e., individual duplicate reports point to one master report independently). In some cases, the relationships are transitive: master report (M) ⟵ first duplicate (D1) ⟵ second duplicate (D2). We regard them as a group of duplicate reports for the master report and dealt with as pairs of <M, D$*$ >. Note that we cannot collect duplicate bug reports for ZXing as shown in Table 1 because its ITS for old reports is no longer available.

In this work, to avoid introducing bias related to data quality, we do not leverage any of the advanced approaches [17, 36, 40, 41] to detect duplicate reports. Instead, we use the link explicitly specified in each bug report. This method may miss potential duplicate bug reports but there might be no false positives. Overall, we collected 807 duplicate report pairs[6] for the new subjects and 136 pairs for the old subjects. Note that duplicate reports often are not attached with fixed files since those reports tend to be classified as duplicates before linking source files to change. Thus, we assume, when using for our experiments, those duplicate reports have the same set of files that the master reports have changed to fix the bugs.

3.3 Bug Linking

To build experimental ground truth we must link every fixed bug report with the associated files where the fix is performed. The correct link information is crucial for evaluating bug localization techniques since the techniques rely on the link information in their performance evaluation. The researchers of most IRBL studies also collected the link information between bug reports and source code commits to evaluate the techniques.

(a) Example of link information (commit ID of a project repository) shown in a bug report.



(b) Example of link information (bug report ID of an issue tracking system) shown in a commit log message.

Figure 1: Examples of two bug linking methods.

Although there exist other more advanced approaches to bug linking (e.g., ReLink [49], MLink [31], and RCLinker [21]), we do not consider them to avoid their performance limitations [6]. Wu et al. [49] reported relatively high precision results for half of the subjects, but their technique showed lower precision (0.682–0.858) for another half. Although MLink [31] achieved a higher precision than ReLink, its evaluation was performed a limited number/size of subjects. This limitation is also discussed in [21] with an evaluation of additional subjects; its precision values are ranged from 0.3 to 0.785 (average: 0.564). RCLinker [21] achieves 0.509 precision on average. Due to the low precision, we cannot adopt the bug linking techniques in our study. The six IRBL techniques used in our study did not leverage the above approaches to avoid many false positives.

In this study, we only use explicit links in bug reports and commit log messages since our goal is to avoid false positives (a link between a report and file but they are not actually relevant) and to allow false negatives (missing links). Developers often put link information in bug reports and commit logs to keep track of bugs and corresponding source code changes [20]. They use explicit IDs to identify bug reports and code commits. Figure 1a shows an example of a commit ID recorded in a bug report. Most issue tracking systems provide a feature to explicitly designate commit ID (with a hyperlink). In case of commit logs, developers manually write bug report ID in its message in a specific format as shown in Figure 1b.

We extracted link information in bug reports, which is explicitly provided by an issue tracking system. The subjects listed in Table 1 are all managed by Jira. Thus, we collected bug reports written in Jira's format and we identified commit IDs after parsing them. If a bug report contains a designated commit ID, we link the source code files (in this study, "`*.java`" files) to the bug report. When multiple commit IDs are recorded in a single bug report, we use the latest commit ID to link source code files.

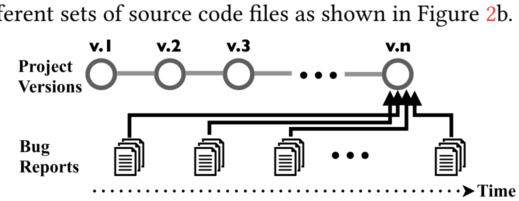
When identifying bug report IDs in a commit log, we use a regular expression. Bug report IDs of the subjects are often recorded in a format of "`PROJECT-###`", which can be detected by a regular expression. If there are multiple IDs, we use the first occurrence as the bug report to link source code files in the commit.

[6]The number of pairs and duplicate reports are different since some duplicate reports can be grouped by a master report as explained in Section 3.2.

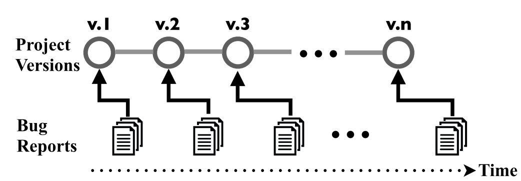
3.4 Version Matching

Bug reports are submitted by users in reference to specific project versions. In most cases, the target version of the bug is the most recently released version at the time of submission. When attempting to localize potential buggy files relevant to a given bug report, it is necessary to clearly define a search space of the project files. If version information is not correctly specified, the result of a bug localization technique becomes useless.

In this study, we apply two different strategies of version matching: *single version matching* vs. *multiple version matching*. In the single version matching strategy, a bug localization technique assumes that the search space of potential source code files is the latest version of a software project as shown in Figure 2a. This strategy makes the search space simple. Thus, most bug localization studies have used this strategy in their evaluation. The work of the six techniques adopted in this study also used this strategy in their own evaluation. With multiple version matching, a bug localization technique constructs a set of files of the version specified in a bug report as the search space. Thus, different bug reports correspond to different sets of source code files as shown in Figure 2b.



(a) Single version matching strategy.



(b) Multiple version matching strategy.

Figure 2: Two version matching strategies used in this study.

Since previous studies did not implement multiple version matching, we perform the following procedure to determine which version corresponds to each report: (1) we collect all tagged version releases for each subject, then (2) we check, for each bug report, that it specifies the release version(s) that are concerned by this bug. When several versions are listed, we consider the oldest one. If there is no version specified in a bug report, we ignore the report. Note that the numbers in "# Bug reports" in Table 1 indicate the number of bug reports with explicit version information. For old subjects, version matching is not applied since many major releases associated with reports used in previous studies are no longer available.

3.5 Test File Inclusion

This study uses two different strategies for search space construction: (1) with test code files and (2) without test files. Most IRBL techniques select the first strategy; a bug localization technique scans all source code files (e.g., "∗.java") in a target project to create a search space. Among the six techniques adopted in this study, only Locus [45] does not include test code files. The technique excludes test code files since the authors assumed that test files are not the target of bug localization. Specifically, the authors stated that the inclusion of test files may cause bias or noise since some test files contain specific bug identifiers and they are created after the bug is fixed. Note that this situation happens only when using single version matching.

To prepare the search space without test code files, we use the following procedure: (1) filter out files if its path contains ". . . /test/. . . " or ". . . /tests/. . . ", and (2) exclude source code files if its file name ends with ". . . Test.java". Although this procedure may cause false positives (i.e., non-test files can be excluded) or false negatives (i.e., test files can remain in the search space), most projects in the subjects listed in Table 1 follow the path and file naming rules and the above procedure can be effective.

4 ANALYSIS RESULTS

4.1 Baseline Performance

We apply each considered IRBL technique to every subject listed in Table 1, and check the output ranked list of source code files identified as potential bug locations against the collected ground truth dataset previously collected from the subjects. As described in Section 3.1, we use the default parameters specified in the original work of the six IRBL techniques.

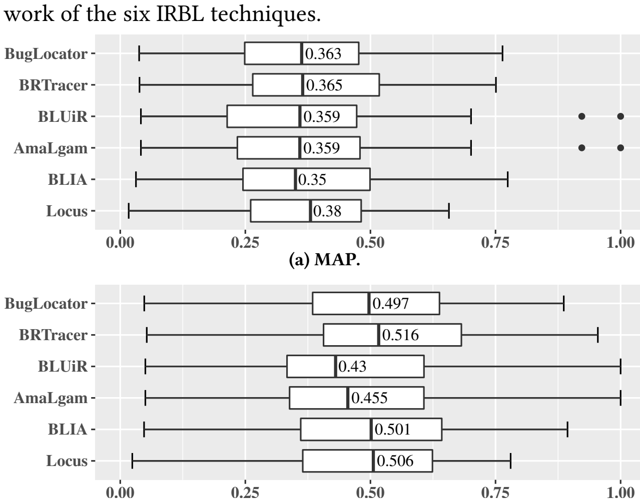


Figure 3: Distribution of MAP/MRR values of subjects for each technique: All (old/new) subjects listed in Table 1 with single version matching and test files included.

Note that the performance values shown in Figure 3 are measured with the following setting: (1) all subjects (old + new) listed in Table 1, (2) single version matching strategy, and (3) including test code files. This setting is similar to the configurations used in the evaluation of the six techniques described in Section 3.1 (cf., Locus does not include test files by default).

We measured the performance (MAP and MRR) by applying each IRBL technique to each subject. Figure 3 shows the overall distributions of MAP and MRR for each technique. MAP values of all techniques are ranged from 0 to 1. While BLUiR and AmaLgam have higher outliers, BLIA has the best top value (0.774 of the **IO** subject). The average values are 0.37, 0.39, 0.37, 0.38, 0.36, and 0.36, respectively (in sequence of BugLocator, BRTracer, BLUiR, AmaLgam, BLIA, and Locus). MRR (Figure 3b) values are also ranged from 0 to 1. The average values are 0.50, 0.52, 0.47, 0.48, 0.49, and 0.47, respectively (with the same sequence of MAP's average values).

In spite of small differences observed above, none of the six techniques substantially outperforms the others, all presenting a tight range of MAP and MRR values; the median values of MAP and MRR are ranged as 0.350 – 0.380 and 0.430 – 0.516, respectively.

From the perspective of practice, users of IRBL techniques can expect a similar performance by using any of the six techniques. Although each technique yields better performance for some projects (MAP: 0.6 — 0.75 and MRR: 0.75 – 1.0), it is not feasible to figure out whether an IRBL technique works better for a specific project without a prior knowledge. This encourages to characterize project characteristics and investigate the sensitivity of IRBL techniques on the characteristics.

In addition, the overall MAP and MRR values suggest that the field of IR-based bug localization still has much room for improvement from the research perspective. Despite recent efforts in bug localization, 35 – 50% precision and recall values (even for file-level) might not acceptable to practitioners. Further investigation should include (1) the characteristics of those projects for which the IRBL techniques ineffectively perform, (2) the acceptable level of performance to users, and (3) effective granularity of localization.

On the need for a further analysis

There are no significant differences, in terms of performance, among the six investigated IRBL techniques. A thorough analysis of the limitations can reveal various opportunities for research directions for improving the state-of-the-art.

4.2 RQ1: Subject Groups

To compare the performance of the techniques between old and new subject groups, we compute MAP and MRR values for each subject. Our results are summarized in two different ways: (1) Table 2 summarizes the results of each technique assuming all the bug reports are aggregated into a single (virtual) project and (2) Table 3 shows individual performance values each pair of project and technique. In the single virtual evaluation setting, we compute MAP and MRR values assuming all bug reports are in a single project. In the latter setting, the values are computed by each project. Therefore, the average MAP/MRR values in Table 3 are different from the values in Table 2. In Table 2, we use Mann-Whitney U test [26] to find out whether the differences are significant. This statistical test is also used for comparing the average of the following experiments. In Table 3, the highest MAP and MRR values for each subject across the six techniques are highlighted by background color and bold-face font.

Table 2: Summary of MAP/MRR of IRBL techniques for old and new subjects in Table 1 (aggregated results with single version matching and test files included).

Technique	Old (Single Ver.)		New (Single Ver.)	
	MAP	MRR	MAP	MRR
BugLocator	0.2692	0.3985	↗ **0.3052****	↗ **0.4223****
BRTracer	0.2645	0.3664	↗ **0.3330****	↗ **0.4690****
BLUiR	0.3102	0.4556	↘ **0.2881****	↘ 0.3869
AmaLgam	0.2950	0.4072	↘ **0.2906****	↘ **0.3899****
BLIA	0.2935	0.4242	↗ **0.3014****	↘ **0.4155****
Locus	0.2641	0.3399	↗ **0.3289****	↗ **0.4430****

*: p-value < 0.05, **: p-value < 0.01, ↗: increased, ↘: decreased

Note that we use the single version matching strategy (see Section 3.4) and include test code files in the search space (see Section 3.5) in this experiment for RQ1. These experiment strategies are common to the original studies of six IRBL techniques we used in our study except for Locus [45]. This setting might be vulnerable to recommending files that did not exist when a given bug report was submitted.

As shown in Table 2, the six techniques differently perform between old and new subjects at the level of bug reports. In cases of BLUiR and AmaLgam, both MAP and MRR values for new subjects are lower than those of old subjects. BLIA's MRR is also lower for new subjects than those of old subjects. For other cases, MAP/MRR values are higher for new subjects. BLUiR shows the best MAP/MRR on average for old subjects while BRTracer achieves the highest MAP/MRR values on average for new subjects for project-level performance (see Table 3). Locus yields the best MAP for 17 individual projects (16 of new and one of old) and BRTracer takes the top MRR values for 21 (20 of new and one of old) individual projects.

Note that the MAP/MRR values for old subjects in Table 3 are different from the values reported in each of the six techniques. The differences might be caused by (1) different number of bug reports and (2) different configuration such as version matching and test file inclusion. The studies of the six techniques have used different numbers of reports even for the same subject. While most of the previous studies adopted single version matching, each study has used different versions of each subject for evaluation; this can be another reason for the different results.

The results imply that IRBL techniques are actually **not overfitted** to old subjects. While the previous studies tend to (re-)use the same outdated subjects, we find that the community should turn to up-to-date subjects to describe the performance that developers could find relevant.

One of the potential reasons of better performance of the new subjects is that developers tend to write source code more with user languages; Our conjecture is that the developers of recent projects may use user-friendly words for identifiers in source code. Thus, token distributions of both bug reports and source code could be more similar.

On the use of old vs. new subjects

IRBL techniques generally yield better performance on recent subjects. To estimate the actual performance that is reached by state-of-the-art approaches, we recommend that researchers should use up-to-date subjects.

Table 3: MAP/MRR for each subject listed in Table 1 (project-wise results with single version matching and test files included).

Subject	BugLocator		BRTracer		BLUiR		AmaLgam		BLIA		Locus	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
New subjects												
CAMEL	0.3235	0.4621	0.3646	0.5270	0.3005	0.4188	0.3032	0.4210	0.3097	0.4451	**0.3986**	**0.5571**
HBASE	0.2993	0.4168	**0.3463**	**0.4884**	0.2818	0.3938	0.2820	0.3942	0.3094	0.4258	0.3084	0.4059
HIVE	0.2693	0.3670	0.3178	0.4521	0.2769	0.3914	0.2772	0.3916	0.2412	0.3312	**0.3310**	**0.4580**
CODEC	0.6227	0.8341	0.6333	0.8199	**0.7011**	0.8625	0.7011	0.8625	0.6914	**0.8829**	0.3519	0.4087
COLLECTIONS	0.2318	0.3327	0.2319	0.3457	0.2174	0.2659	0.2191	0.2659	0.2493	**0.3606**	**0.2670**	0.3090
COMPRESS	0.5637	0.7545	0.5747	**0.7976**	0.4874	0.6382	0.4816	0.6345	0.5797	0.7822	**0.5872**	0.7801
CONFIGURATION	0.0378	0.0480	0.0385	**0.0530**	**0.0413**	0.0502	0.0413	0.0502	0.0345	0.0476	0.0299	0.0430
CRYPTO	0.1622	0.2665	0.1711	0.3044	**0.1982**	**0.3974**	0.1982	0.3974	0.1588	0.3616	0.1118	0.3421
IO	0.7640	0.8574	0.7508	**0.8809**	0.6797	0.7332	0.6784	0.7309	**0.7744**	0.8489	0.4053	0.4322
LANG	0.5371	**0.6446**	0.5426	0.6441	0.5327	0.5783	0.5367	0.5810	0.5349	0.6423	**0.5810**	0.6181
MATH	0.1563	0.2173	0.1586	0.2274	0.1952	0.2413	**0.2122**	**0.2627**	0.1765	0.2394	0.1895	0.2251
WEAVER	0.6212	0.6666	0.6331	**0.7500**	**0.6637**	0.6666	0.6637	0.6666	0.5695	0.6666	0.4996	0.5500
CSV	0.6104	0.6677	0.6108	0.6784	0.6075	0.6369	0.6313	0.6845	0.4048	0.6429	**0.6554**	**0.6970**
ENTESB	0.0559	0.0511	0.0542	0.0624	**0.0652**	**0.0775**	0.0652	0.0775	0.0314	0.0638	0.0555	0.0704
JBMETA	0.2442	**0.4387**	0.2351	0.4312	0.1805	0.3233	0.1832	0.3258	0.2078	0.3698	**0.2534**	0.3639
ELY	0.0587	0.1506	0.0668	0.1587	0.1098	**0.1962**	0.1098	0.1962	0.0939	0.1667	**0.1260**	0.1667
WFARQ	0.5000	0.5000	0.3333	0.3333	**1.0000**	**1.0000**	1.0000	1.0000	0.1111	0.1111	0.5000	0.5000
WFCORE	0.3202	0.4444	0.3326	**0.4687**	0.2552	0.3392	0.2565	0.3402	0.2830	0.3834	**0.3607**	0.4607
WFLY	0.2283	0.3196	**0.2572**	**0.3623**	0.2099	0.2972	0.2104	0.2982	0.2118	0.2994	0.2562	0.3500
WFMP	0.4534	0.5833	0.2675	0.2398	**0.6818**	**0.8333**	0.6818	0.8333	0.5000	0.6667	0.6226	0.7778
SWARM	0.2936	0.3852	0.3326	**0.4317**	**0.3412**	0.3960	0.3412	0.3960	0.2708	0.3608	0.2651	0.3656
AMQP	0.4426	0.6205	**0.4775**	**0.6754**	0.4196	0.5691	0.4210	0.5694	0.4626	0.6661	0.4533	0.6229
ANDROID	0.3626	0.5121	0.3536	**0.5904**	**0.3761**	0.4747	0.3761	0.4747	0.3138	0.5076	0.0796	0.0698
BATCH	0.3186	0.4848	0.3284	0.5023	0.2805	0.3941	0.2933	0.4124	0.3158	0.4735	**0.3664**	**0.5449**
BATCHADM	0.3218	0.4325	0.3479	0.5011	0.3657	0.4806	0.3657	0.4806	0.4313	0.5724	**0.4322**	**0.6252**
DATACMNS	0.4565	0.5916	0.4683	0.6429	0.4581	0.5632	0.4581	0.5632	0.5133	**0.6670**	**0.5244**	0.6482
DATAGRAPH	0.1519	0.2384	0.1592	0.2497	0.1714	**0.2788**	**0.1718**	0.2788	0.1426	0.2577	0.1609	0.2578
DATAJPA	0.4822	0.6649	0.4892	0.6854	0.4769	0.6170	0.4767	0.6170	**0.5311**	**0.7098**	0.4769	0.6329
DATAMONGO	0.4582	0.6344	0.5095	**0.6965**	0.4519	0.5769	0.4519	0.5769	**0.5212**	0.6807	0.4861	0.6322
DATAREDIS	0.5353	0.7627	0.5561	0.8081	0.5801	0.7626	**0.5813**	0.7637	0.5717	**0.8082**	0.4999	0.7259
DATAREST	0.3492	0.5458	**0.3939**	**0.6372**	0.3550	0.5125	0.3584	0.5162	0.3651	0.5730	0.3802	0.5974
LDAP	0.4401	0.6344	**0.4875**	**0.7197**	0.4681	0.6251	0.4681	0.6251	0.4824	0.6665	0.3857	0.5058
MOBILE	0.6909	0.8864	0.7116	0.9545	**0.9224**	**1.0000**	0.9224	1.0000	0.7285	0.8939	0.5042	0.5862
ROO	0.1164	0.1628	**0.1293**	**0.1821**	0.0910	0.1283	0.0928	0.1297	0.1001	0.1422	0.1208	0.1811
SEC	0.3209	0.4237	0.3368	0.4438	0.3087	0.3736	0.3133	0.3788	**0.3502**	**0.4496**	0.3145	0.3857
SECOAUTH	0.1983	0.3659	**0.2128**	**0.3965**	0.1381	0.2620	0.1430	0.2714	0.1891	0.3522	0.1990	0.3683
SGF	0.4173	0.6546	0.4223	0.6850	0.3591	0.5973	0.3589	0.5970	0.3682	0.6174	**0.4359**	**0.7245**
SHDP	0.4433	0.6279	0.4652	**0.6734**	0.3899	0.5184	0.3897	0.5184	**0.4654**	0.6222	0.4633	0.5826
SHL	0.2533	0.4037	0.2621	0.4166	0.2827	0.4220	0.2828	0.4221	0.2836	0.4015	**0.3251**	**0.4579**
SOCIAL	0.6110	0.6937	0.5900	0.6726	0.1979	0.2245	0.2496	0.3000	0.5285	0.5689	**0.6569**	**0.7029**
SOCIALFB	0.5541	0.6416	**0.6156**	**0.7401**	0.4818	0.6167	0.4818	0.6167	0.4064	0.5301	0.5382	0.6929
SOCIALLI	0.4711	0.6250	**0.6384**	**0.7083**	0.3929	0.3958	0.4504	0.4166	0.2989	0.3208	0.4081	0.6875
SOCIALTW	**0.7382**	**0.7937**	0.6750	0.7292	0.3814	0.4271	0.5014	0.5833	0.5594	0.6188	0.5456	0.6250
SPR	0.3074	0.4684	**0.3377**	**0.5165**	0.2061	0.3284	0.2182	0.3386	0.2878	0.4319	0.0169	0.0241
SWF	0.3812	0.4758	0.3974	0.5060	0.3647	0.4579	0.3613	0.4548	0.4038	0.5015	**0.4384**	**0.5502**
SWS	0.4002	0.5400	**0.4211**	**0.5872**	0.3811	0.4886	0.3811	0.4886	0.3969	0.5456	0.4177	0.5680
Average	0.3821	0.5064	**0.3922**	**0.5299**	0.3767	0.4746	0.3836	0.4827	0.3644	0.4930	0.3649	0.4757
Old subjects												
AspectJ	0.2148	0.3857	0.1428	0.2587	0.2250	0.4161	0.2023	0.3382	**0.2505**	**0.4185**	0.1605	0.2186
ZXing	0.3306	0.3837	0.4598	0.5626	0.3937	0.4219	0.4596	0.5626	**0.4984**	**0.5752**	0.4358	0.5274
PDE	0.3757	0.4971	0.3330	0.4304	**0.3884**	**0.5382**	0.3334	0.4308	0.3376	0.4714	0.3711	0.5236
JDT	0.1697	0.2699	0.2491	0.3382	0.2774	0.3832	0.2491	0.3382	0.1742	0.2310	**0.3594**	**0.4542**
SWT	0.4458	0.5016	**0.5528**	**0.6286**	0.5257	0.5967	0.5528	0.6284	0.4639	0.5666	0.3303	0.3812
Average	0.3073	0.4076	0.3475	0.4437	**0.3620**	**0.4712**	0.3594	0.4596	0.3449	0.4525	0.3314	0.4210

For each subject, the highest MAP and MRR values are highlighted by blue (**n.nnnn**) and green (**n.nnnn**) background, respectively, with bold-face font.

4.3 RQ2: Impact of Matching Bug Report with Code Version

We utilize the version matching strategies described in Section 3.4 to investigate the importance of using a consistent code version with regards to a given bug report in order to accurately locate buggy files. We applied version matching only to new subjects, unfortunately, due to the limited availability of version releases in old subjects.

Table 4 provides the summarized results for two different strategies: single and multiple version matching. All six IRBL techniques show higher MAP and MRR values when applying multiple version matching. The performance improvements are even statistically significant for all techniques. The improvement of MAP is ranged from 0.0661 (BugLocator) to 0.0928 (Locus) and that of MRR is from 0.0836 (BRTracer) to 0.1084 (Locus).

Table 4: Summary of comparison between single vs. multiple version matching strategies (aggregated results with test files included).

Technique	New (Single Ver.)		New (Multiple Ver.)	
	MAP	MRR	MAP	MRR
BugLocator	0.3052	0.4223	$\nearrow$ **0.3713****	$\nearrow$ **0.5075****
BRTracer	0.3330	0.4690	$\nearrow$ **0.3992****	$\nearrow$ **0.5526****
BLUiR	0.2881	0.3869	$\nearrow$ **0.3623****	$\nearrow$ **0.4802****
AmaLgam	0.2906	0.3899	$\nearrow$ **0.3657****	$\nearrow$ **0.4840****
BLIA	0.3014	0.4155	$\nearrow$ **0.3777****	$\nearrow$ **0.5124****
Locus	0.3289	0.4430	$\nearrow$ **0.4217****	$\nearrow$ **0.5514****

$*$: p-value < 0.05, $**$: p-value < 0.01, $\nearrow$: increased, $\searrow$: decreased

Table 5 shows project-wise performance results after applying version matching. Compared to Table 3, Locus shows the best MAP on average (0.4368) and takes the highest MAP for the most number (=24) of projects. BRTracer yields the highest MRR values for the most number (=13) of projects and the best average MRR value all over the projects.

The results can be interpreted in different ways. First, single version matching often makes the search space larger than an actual version corresponding to a target report since the matching strategy uses the latest version as the search space; Developers tend to add more files rather than removing them as a program is updated. Second, target files can be missing in the latest version. Since some files can be removed from a project after revisions, the target files that IRBL techniques need to identify might be missing in the latest version when using single version matching.

The results of this study imply that the evaluation/execution of IRBL techniques should apply multiple version matching. Existing studies have evaluated their techniques by using single version matching because it makes the evaluation simpler. However, this degrades the achievement of the IRBL techniques. In addition to the effect of evaluation, multiple version matching is actually close to the real application setting.

On the importance of version matching

Our experiments have shown that when matching the bug report with its code version, IRBL techniques are most effective. IRBL techniques should consider exploiting version metadata from bug report and select appropriate code base for attempting to localize the bug.

4.4 RQ3: Test File Inclusion

This section reports the performance differences between two strategies of test file inclusion. Following the procedure of identifying test code files described in Section 3.5, we filter out test files from both of the search space and a set of target files that IRBL techniques need to find. Then, we run each IRBL techniques on the subjects with the multiple version matching strategy. The aggregated results of MAP/MRR are summarized in Table 6 while project-wise performance is listed in Table 8.

As shown in Table 6, excluding test code files results in a negative effect on the overall performance of IRBL techniques. For a half (BugLocator, BRTracer, and BLIA) of the techniques, it slightly improved MAP (by 0.098, 0.149, and 0.125, respectively) while the values are decreased for another half (BLUiR, AmaLgam, and Locus) by -0.02, -0.024, and -0.071, respectively. The test exclusion strategy constantly makes a negative effect for MRR of all IRBL techniques, ranged from -0.0396 to -0.0512.

Table 5: MAP/MRR with multiple version matching (project-wise results with test files included).

Subject	BugLocator		BRTracer		BLUiR		AmaLgam		BLIA		Locus	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
New subjects												
CAMEL	0.3585	0.5058	0.3960	0.5641	0.3487	0.4775	0.3525	0.4824	0.3479	0.4857	**0.4277**	**0.5900**
HBASE	0.3626	0.4999	0.4004	0.5548	0.3533	0.4891	0.3531	0.4890	0.3823	0.5219	**0.4322**	**0.5809**
HIVE	0.3048	0.4140	0.3517	**0.4870**	0.3177	0.4402	0.3181	0.4403	0.2888	0.3884	**0.3546**	0.4822
CODEC	0.6390	0.8483	0.6237	0.7942	0.7488	0.9028	0.7488	0.9028	0.6734	0.8710	**0.7516**	**0.9286**
COLLECTIONS	0.4443	0.6312	0.5236	0.6998	0.4057	0.4713	0.4057	0.4713	0.5692	0.7288	**0.6602**	**0.7637**
COMPRESS	0.5414	0.7377	0.5484	0.7692	0.4839	0.6249	0.4794	0.6207	0.5791	0.7961	**0.6109**	**0.7985**
CONFIGURATION	0.5174	0.7164	0.5103	0.7376	0.5063	0.6234	0.5143	0.6289	0.5750	**0.7776**	**0.6095**	0.7379
CRYPTO	0.1628	0.2665	0.1708	0.3044	**0.1982**	**0.3974**	0.1982	0.3974	0.1588	0.3616	0.1118	0.3421
IO	0.7110	0.8214	0.7152	**0.8570**	0.6694	0.7103	0.6693	0.7101	0.7629	0.8548	**0.7943**	0.8459
LANG	0.5576	0.6749	0.5648	0.6771	0.5674	0.6197	0.5714	0.6221	0.5988	**0.7193**	**0.6737**	0.7143
MATH	0.4227	0.5704	0.4459	0.6304	0.4834	0.6058	**0.4965**	0.6209	0.4696	**0.6351**	0.4756	0.5776
WEAVER	0.6212	0.6666	0.6331	**0.7500**	**0.6637**	0.6666	0.6637	0.6666	0.5695	0.6666	0.3580	0.2917
CSV	0.6017	0.6672	0.5813	0.6448	0.6052	0.6369	0.6290	0.6845	0.3750	0.5357	**0.6554**	**0.6970**
ENTESB	0.0999	0.1131	0.0875	0.1189	0.0949	0.1212	**0.1546**	**0.1919**	0.0539	0.0944	0.1178	0.1587
JBMETA	0.2272	0.4005	0.2332	**0.4057**	0.1685	0.2940	0.1706	0.2960	0.2078	0.3673	**0.2717**	0.3977
ELY	0.0625	0.1540	0.0673	0.1587	0.1098	**0.1962**	0.1098	0.1962	0.0939	0.1667	**0.1270**	0.1733
WFARQ	0.5000	0.5000	0.3333	0.3333	**1.0000**	**1.0000**	1.0000	1.0000	0.1111	0.1111	0.5000	0.5000
WFCORE	0.3225	0.4407	0.3353	**0.4658**	0.2693	0.3566	0.2706	0.3578	0.2826	0.3943	**0.3633**	0.4603
WFLY	0.3036	0.4174	0.3337	0.4614	0.2698	0.3751	0.2706	0.3762	0.2624	0.3694	**0.3542**	**0.4753**
WFMP	0.4534	0.5833	0.2675	0.2398	**0.8485**	**1.0000**	0.8485	1.0000	0.5000	0.6667	0.6226	0.7778
SWARM	0.2857	0.3510	**0.2967**	0.3949	0.2778	0.3392	0.2778	0.3392	0.2522	0.3426	0.2849	**0.4013**
AMQP	0.4136	0.6288	0.4369	0.6608	0.4275	0.6049	0.4306	0.6064	0.4532	**0.6806**	**0.4864**	0.6591
ANDROID	0.4016	0.6582	0.3238	0.5907	0.3421	0.4444	0.3421	0.4444	0.3195	0.5091	**0.5405**	**0.6593**
BATCH	0.3535	0.5379	0.3685	0.5643	0.3738	0.5251	0.3808	0.5311	0.3845	0.5674	**0.4234**	**0.5969**
BATCHADM	0.3825	0.4831	0.4095	0.5678	0.3898	0.5017	0.3898	0.5017	**0.4602**	0.5990	0.4513	**0.6002**
DATACMNS	0.3649	0.5258	0.3766	0.5634	0.3631	0.4880	0.3630	0.4880	0.4319	**0.6064**	**0.4692**	0.5865
DATAGRAPH	0.0396	0.0676	0.0494	0.0999	0.0352	0.0618	0.0352	0.0618	0.0465	0.0877	**0.0667**	**0.1422**
DATAJPA	0.4022	0.5631	0.4150	0.6125	0.4221	0.5795	0.4236	0.5800	**0.4816**	**0.6824**	0.4106	0.5413
DATAMONGO	0.4106	0.5815	0.4412	0.6218	0.4111	0.5456	0.4111	0.5456	0.4724	**0.6487**	**0.4747**	0.6173
DATAREDIS	0.5255	0.7505	0.5478	**0.8000**	0.5523	0.7557	0.5559	0.7593	**0.5571**	0.7689	0.4904	0.7043
DATAREST	0.3377	0.5431	**0.3600**	**0.5781**	0.3297	0.5005	0.3335	0.5043	0.3511	0.5657	0.3094	0.4889
LDAP	0.4697	0.6441	**0.5394**	**0.7521**	0.5017	0.6390	0.5017	0.6390	0.4948	0.6490	0.3821	0.4901
MOBILE	0.6697	0.8864	0.7057	0.9545	**0.9264**	**1.0000**	0.9264	1.0000	0.7316	0.8939	0.4519	0.5732
ROO	0.4111	0.5179	**0.4436**	**0.5646**	0.3874	0.4906	0.3970	0.5000	0.4168	0.5230	0.4019	0.4957
SEC	0.4606	0.5855	0.4822	0.6185	0.4421	0.5249	0.4480	0.5332	**0.5188**	**0.6514**	0.5126	0.6024
SECOAUTH	**0.3300**	0.4525	0.3233	0.4410	0.2975	0.3825	0.3100	0.4011	0.3153	0.4480	0.3244	**0.4591**
SGF	0.3876	**0.6179**	0.3717	0.6112	**0.3899**	0.5986	0.3898	0.5985	0.3808	0.6000	0.3542	0.5972
SHDP	0.4127	0.5859	0.4433	**0.6302**	0.3500	0.4682	0.3500	0.4682	0.4341	0.5799	**0.4501**	0.5774
SHL	0.2802	0.4554	0.2363	0.4226	0.3238	0.4791	**0.3239**	0.4792	0.3160	**0.5152**	0.2100	0.2907
SOCIAL	0.6108	0.6608	0.6237	0.6733	0.4422	0.5092	0.4342	0.4764	0.5916	0.6667	**0.6633**	**0.7029**
SOCIALFB	0.6040	0.6744	**0.6194**	**0.7400**	0.5003	0.5874	0.4804	0.5930	0.4610	0.5103	0.5418	0.6595
SOCIALLI	0.4711	0.6250	**0.6384**	**0.7083**	0.3929	0.3958	0.4504	0.4166	0.2989	0.3208	0.4081	0.6875
SOCIALTW	0.4917	0.6198	0.4941	0.6875	0.5789	0.6667	**0.6622**	**0.7917**	0.3115	0.4479	0.5456	0.6250
SPR	0.3372	0.5056	**0.3708**	**0.5441**	0.2380	0.3930	0.2483	0.4009	0.2985	0.4347	0.3573	0.5203
SWF	0.3708	0.4828	0.3912	0.5208	0.3706	0.4808	0.3739	0.4830	0.3916	0.5027	**0.4415**	**0.5452**
SWS	0.3515	0.4761	0.3504	0.4904	0.3391	0.4558	0.3391	0.4558	0.3579	**0.5017**	**0.3661**	0.4870
Average	0.4085	0.4085	0.4170	**0.4170**	0.4243	0.4243	0.4305	0.4305	0.3998	0.3998	**0.4368**	0.4368

For each subject, the highest MAP and MRR values are highlighted by blue (n.nnnn) and green (n.nnnn) background, respectively, with bold-face font.

Project-wise results (Table 8) also show the performance degradation. While Locus achieves the best MAP on average in the previous experiment (see Table 5), BRTracer shows the best MAP and MRR values on average as shown in Table 8. The number of best-performing projects is also decreased for Locus (in case of MAP, from 24 to 16 projects).

Table 6: Summary of comparison between test file inclusion strategies (aggregated results with multiple version matching).

Technique	Test files included		Test files excluded	
	MAP	MRR	MAP	MRR
BugLocator	0.3713	0.5075	$\nearrow$ **0.3811****	$\searrow$ **0.4647****
BRTracer	0.3992	0.5526	$\nearrow$ 0.4141	$\searrow$ **0.5090****
BLUiR	0.3623	0.4802	$\searrow$ **0.3603****	$\searrow$ **0.4385****
AmaLgam	0.3657	0.4840	$\searrow$ **0.3633****	$\searrow$ **0.4420****
BLIA	0.3777	0.5124	$\nearrow$ **0.3902***	$\searrow$ **0.4728****
Locus	0.4217	0.5514	$\searrow$ **0.4146****	$\searrow$ **0.5002****

$*$: p-value < 0.05, $**$: p-value < 0.01, $\nearrow$: increased, $\searrow$: decreased

There might be several reasons for the above results. First, test code files would contain many common words written in bug reports. It is a common conjecture that a smaller search space would produce better recommendation results (e.g., excluding test files).

However, it turns out that test files can be ranked in higher places and may help bug localization even though they can make the search space larger. This indicates that test files contain many tokens appearing in bug reports. Second, more vulnerable files are tested more. Developers tend to write test cases earlier and to change frequently if a file is vulnerable to bugs. Thus, test files for the file would contain tokens well-describing some specific symptoms and causes; IRBL techniques can identify them easily. Third, some test files include test cases of multiple classes and methods. This might happen when developers write a monolithic test code. Thus, it is necessary to change the test files for any bug is fixed.

Test code files should be included in the search space of IRBL techniques. Those files may contain more text tokens relevant to the user's perspective (i.e., tokens in bug reports) than other code files. Since test files are eventually connected to production files, including test files would help the bug localization process. Note that the evaluation of Locus filtered out test files [45] since they assumed that those test files can cause bias and noise in the results of their tool. The bias is also related to the version matching strategy as the single version matching strategy can result in the inclusion of files created after a bug is reported. However, including test files does not interfere bug localization if it is with multiple (and correct) version matching. In fact, it slightly improves the performance of IRBL techniques.

On the impact of test file inclusion

With a correct version matching strategy, including test files does not make bias or noise in the evaluation of IRBL techniques.

Table 7: Performance with/without duplicate reports.

Tool	Master		Duplicate		Master + Duplicate	
	MAP	MRR	MAP	MRR	MAP	MRR
BugLocator	0.3503	0.5051	$\searrow$ 0.3259	$\searrow$ 0.4667	$\searrow$ 0.3502	$\nearrow$ 0.5249
BRTracer	0.3852	0.5508	$\searrow$ 0.3776	$\searrow$ 0.5430	$\searrow$ 0.3787	$\nearrow$ 0.5692
BLUiR	0.3159	0.4540	$\searrow$ **0.2804***	$\searrow$ **0.4192***	$\nearrow$ 0.3325	$\nearrow$ 0.4728
AmaLgam	0.3202	0.4581	$\searrow$ **0.2829****	$\searrow$ **0.4223***	$\nearrow$ 0.3327	$\nearrow$ 0.4725
BLIA	0.3518	0.4915	$\searrow$ **0.3231***	$\searrow$ 0.4537	$\nearrow$ 0.3577	$\nearrow$ 0.5041
Locus	0.2915	0.4707	$\searrow$ 0.2871	$\nearrow$ 0.4724	$\nearrow$ 0.3042	$\nearrow$ 0.5021

$*$: p-value < 0.05, $**$: p-value < 0.01, $\nearrow$: increased, $\searrow$: decreased

4.5 RQ4: Potential Contribution of Duplicate Bug Reports

Considering collected $< M, D >$ pairs (cf. Section 3.2) in our subjects, we compute MAP and MRR values of three sets of reports: (1) master reports of $< M, D >$ pairs, (2) duplicate reports of the pairs, and (3) master+duplicate reports (*M+D*) by applying IRBL techniques. To build *M+D*, we concatenate the text of master and duplicate reports in sequence for each pair.

Table 7 shows the results. Note that the performance results in the first column (for *Master*) are different from overall performance found initially for fixed bug reports (cf. Table 2) because, in this experiment, we only consider master reports that have duplicates.

All six IRBL techniques produced less MAP and MRR values for master or duplicate reports than when combining master and duplicate reports. As shown in the rightmost column in Table 7, MAP and MRR values are increased except for MAP of BugLocator and BRTracer. The difference of MAP and MRR are ranged from -0.007 to +0.017 and +0.013 to +0.031, respectively, although the differences are not statistically significant. This finding is consistent with the general study of Bettenburg et al. [3], which reported that duplicate bug reports often provide useful information (on average 40% more than master reports). On the other hand, we observe cases where duplicate reports actually yield degraded performances (with BLUiR and AmaLgam) where differences with master reports are statistically significant.

Table 8: MAP/MRR after excluding test files from the search space (project-wise results with multiple version matching).

Subject	BugLocator		BRTracer		BLUiR		AmaLgam		BLIA		Locus	
	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
New subjects												
CAMEL	0.3706	0.4658	0.4158	0.5280	0.3682	0.4511	0.3716	0.4550	0.3665	0.4566	**0.4501**	**0.5524**
HBASE	0.3453	0.4433	0.3796	0.4918	0.3303	0.4284	0.3299	0.4280	0.3561	0.4576	**0.3952**	**0.5025**
HIVE	0.2893	0.3848	0.3365	**0.4548**	0.3050	0.4098	0.3054	0.4095	0.2775	0.3627	**0.3432**	0.4497
CODEC	**0.7060**	0.8230	0.6704	0.7554	0.7030	0.8016	0.7030	0.8016	0.7019	**0.8241**	0.6820	0.7935
COLLECTIONS	0.5846	0.6296	0.6437	0.6933	0.4278	0.4581	0.4278	0.4581	0.6614	**0.7252**	**0.6790**	0.7208
COMPRESS	0.6432	0.7165	0.6685	0.7561	0.5751	0.6139	0.5702	0.6097	**0.7054**	**0.7871**	0.6916	0.7693
CONFIGURATION	0.6369	0.6846	0.6576	0.7088	0.5032	0.5433	0.5116	0.5498	**0.6934**	**0.7558**	0.5511	0.5911
CRYPTO	0.1610	0.2639	0.1692	0.3021	**0.1950**	**0.3911**	0.1950	0.3911	0.1545	0.2951	0.0847	0.1059
IO	0.7757	0.7917	0.7944	**0.8316**	0.6577	0.6680	0.6576	0.6678	**0.7994**	0.8274	0.7299	0.7550
LANG	0.5998	0.6154	0.6099	0.6225	0.5506	0.5601	0.5504	0.5580	**0.6713**	**0.6898**	0.5870	0.6050
MATH	0.4127	0.4851	0.4631	0.5416	0.4649	0.5289	0.4746	0.5410	**0.4833**	**0.5558**	0.3968	0.4508
WEAVER	0.4091	0.4166	0.4404	0.3750	0.5328	**0.6666**	0.5328	0.6666	**0.6041**	0.6666	0.2757	0.2917
CSV	0.6142	0.6434	0.5740	0.6032	0.5714	0.5952	0.5714	0.5952	0.5000	0.5357	**0.6381**	**0.6596**
ENTESB	0.0412	0.0515	0.0234	0.0309	0.0477	0.0548	**0.1080**	**0.1256**	0.0118	0.0301	0.0559	0.0748
JBMETA	0.2237	0.3508	0.2360	**0.3570**	0.1759	0.2499	0.1770	0.2510	0.2111	0.3153	**0.2520**	0.3530
ELY	0.0625	0.1540	0.0673	0.1587	0.1098	**0.1962**	0.1098	0.1962	0.0939	0.1667	**0.1270**	0.1733
WFARQ	0.5000	0.5000	0.3333	0.3333	**1.0000**	**1.0000**	1.0000	1.0000	0.1111	0.1111	0.5000	0.5000
WFCORE	0.3004	0.3888	0.3155	0.4107	0.2533	0.3176	0.2544	0.3187	0.2591	0.3368	**0.3482**	**0.4256**
WFLY	0.2653	0.3483	0.2967	0.3877	0.2481	0.3303	0.2515	0.3360	0.2295	0.3040	**0.3175**	**0.4080**
WFMP	0.4534	0.5833	0.2675	0.2398	**0.8485**	**1.0000**	0.8485	1.0000	0.5000	0.6667	0.6226	0.7778
SWARM	0.2437	0.2996	**0.2923**	**0.3741**	0.2631	0.3082	0.2631	0.3082	0.2363	0.3189	0.2678	0.3590
AMQP	0.4818	0.5751	0.5121	0.6005	0.4876	0.5549	0.4906	0.5564	**0.5153**	**0.6064**	0.4973	0.5745
ANDROID	0.3634	0.4612	0.2811	0.3462	0.2751	0.2883	0.2751	0.2883	0.3634	0.4221	**0.4926**	**0.4775**
BATCH	0.4013	0.4837	**0.4299**	0.5188	0.3784	0.4572	0.3804	0.4610	0.4239	0.5159	0.4293	**0.5286**
BATCHADM	0.4241	0.4589	0.4586	0.5081	0.3518	0.4170	0.3518	0.4170	0.4987	0.5614	**0.5091**	**0.5913**
DATACMNS	0.4218	0.4937	0.4538	0.5287	0.3720	0.4367	0.3719	0.4367	**0.4954**	**0.5692**	0.4850	0.5396
DATAGRAPH	0.0342	0.0503	0.0481	0.0853	0.0298	0.0429	0.0298	0.0429	0.0456	0.0681	**0.0562**	**0.1198**
DATAJPA	0.4544	0.5216	0.4807	0.5644	0.4364	0.5080	0.4378	0.5102	**0.5385**	**0.6331**	0.4230	0.4890
DATAMONGO	0.4391	0.4965	0.4805	0.5379	0.3992	0.4553	0.3992	0.4553	**0.4869**	**0.5545**	0.4799	0.5341
DATAREDIS	0.6440	0.7356	**0.6769**	**0.7873**	0.6583	0.7503	0.6619	0.7539	0.6558	0.7439	0.5936	0.6823
DATAREST	0.4022	0.5184	0.4297	0.5534	0.3643	0.4627	0.3620	0.4596	**0.4438**	**0.5562**	0.3466	0.4521
LDAP	0.5044	0.5782	**0.5702**	**0.6780**	0.4908	0.5644	0.4908	0.5644	0.5128	0.6149	0.4069	0.4520
MOBILE	0.4622	0.5411	0.5455	0.6369	**0.6136**	**0.6727**	0.6136	0.6727	0.4811	0.5379	0.3750	0.4218
ROO	0.4030	0.5049	**0.4360**	**0.5530**	0.3784	0.4787	0.3886	0.4882	0.4085	0.5103	0.3968	0.4870
SEC	0.5175	0.5726	0.5431	0.6049	0.4448	0.4900	0.4505	0.4978	**0.5720**	**0.6333**	0.5313	0.5793
SECOAUTH	0.3438	0.4115	0.3417	0.4078	0.2974	0.3495	0.3013	0.3543	0.3525	0.4292	**0.3586**	**0.4362**
SGF	**0.4242**	**0.5794**	0.4146	0.5642	0.3895	0.5283	0.3894	0.5282	0.4038	0.5571	0.3663	0.5358
SHDP	0.4172	0.5264	0.4556	**0.5570**	0.3337	0.4146	0.3337	0.4146	0.4252	0.5131	**0.4688**	0.5246
SHL	0.3157	0.4493	0.2802	0.4183	0.3389	0.4756	0.3390	0.4757	**0.3864**	**0.5152**	0.2034	0.2800
SOCIAL	0.6363	0.6608	0.6459	0.6733	0.4732	0.5092	0.4571	0.4764	0.6156	0.6667	**0.6807**	**0.7029**
SOCIALFB	0.6002	0.6744	**0.6662**	**0.7400**	0.5031	0.5274	0.4882	0.5330	0.4586	0.4743	0.5106	0.5374
SOCIALLI	0.4330	0.5000	**0.5457**	0.5833	0.2629	0.2708	0.3059	0.2916	0.2270	0.2375	0.4717	**0.6250**
SOCIALTW	0.5323	0.5698	0.5804	0.5804	0.5580	0.5833	**0.6518**	**0.7083**	0.3787	0.3854	0.5729	0.5833
SPR	0.3203	0.4230	**0.3532**	**0.4529**	0.2505	0.3425	0.2594	0.3504	0.2913	0.3760	0.3471	0.4415
SWF	0.3988	0.4576	0.4263	0.4932	0.3924	0.4565	0.3943	0.4586	0.4273	0.4854	**0.4498**	**0.5153**
SWS	0.3782	0.4641	0.3899	0.4849	0.3732	0.4458	0.3732	0.4458	**0.4139**	**0.5011**	0.3863	0.4700
Average	0.4216	0.4216	**0.4370**	**0.4370**	0.4127	0.4127	0.4176	0.4176	0.4228	0.4228	0.4312	0.4312

For each subject, the highest MAP and MRR values are highlighted by blue (n.nnnn) and green (n.nnnn) background, respectively, with bold-face font.

On leveraging duplicate bugs reports

We have found that, in general, reports tagged as duplicate are less relevant for localizing buggy files than the associated master report. Nevertheless, duplicate reports bring along additional tokens which can complement master bug reports and guarantee a minimum level of performance.

5 DISCUSSION

5.1 Execution Time

In practice, IRBL technique adoption is not only influenced by the performance that it can deliver but also the execution time. We measure the execution time of each technique for each subject. Table 9 shows a summary of the execution time of the six techniques; the result of each technique is the aggregated value of all versions

Table 9: Execution time (minutes) for the tools.

Technique	Min	Max	Median	Mean	Total
BugLocator	1	70	10	15	702
BRTracer	1	71	10	15	710
BLUiR	1	209	9	22	1,034
AmaLgam	1	217	9	24	1,095
BLIA	1	208	12	21	944
Locus	1	9,636	13	373	17,163

in the subject. Note that AmaLgam's execution time is computed by adding BLUiR's time and AmaLgam's pure execution time since AmaLgam uses the results of BLUiR.

BugLocator and BRTracer need less time (15 min on average) to execute than the rest. BLUiR, AmaLgam, and BLIA took slightly more time (22, 24, and 21 min on average, respectively) than the first two techniques. Running Locus requires significantly more time (373 min on average). Based on our inspection, the original implementation of Locus has a performance issue due to text concatenation in the middle of its execution. We fixed the issue and confirmed that it can significantly improve the performance. However, we do not report here since the goal of this study is to reproduce the results of the original implementations.

Overall, for all six techniques, the execution time is proportional to the number of bug reports in a subject. BLUiR's analysis time is higher than BugLocator and BRTracer due to its use of Indri [39], an open source IR toolkit, which accounts for most of the execution time. Considering revision history incurs more overhead for AmaLgam, BLIA, and Locus. The low runtime performance of Locus is mainly due to change hunk scanning as the technique leverages hunk-level change information even for file-level bug localization.

5.2 Threats to Validity

External validity: Our study examines only Java subjects as listed in Table 1. However, the same process in the study can be applied to other subjects implemented by another programming language. Another threat to the validity of our study is that our subjects are all based on an open source development model. The practice in the software industry may involve projects with specific characteristics that may be even more suitable (or in contrast unsuitable) for IRBL techniques. The provided replication package should help practitioners validate IRBL techniques in their context.

Internal validity: We use the Mann-Whitney U test [26] to examine statistical significance. This method may, however, present limitations. Nevertheless, the methods are commonly used in the literature to figure out the significance.

6 RELATED WORK

Bug localization techniques. Topic modeling and semantic analysis are common techniques used in IRBL. PROMESIR [34] utilizes Latent Semantic Analysis (LSI) [9] to identify buggy files. Lukins et al. [25] adopted Latent Dirichlet Allocation (LDA) [7] to their approach that models source code topics and showed its effectiveness with a small number of case studies. BugScout [30], on the other hand, builds topic models for both source code and bug reports and compares their distribution to locate files to fix a bug.

Stack traces are regarded as a promising information source in bug localization. Wong et al. proposed a BRTracer [46] which further considers stack traces in similarity scores. Lobster [29] also uses stack traces to compare with code elements in source code files. CrashLocator [48] focuses more on stack traces together with function call graphs.

Other IRBL techniques consider machine learning. Ye et al. [51] proposed a learning-to-rank approach to bug localization based features representing the degree of suspiciousness. Kim et al. [18] dealt with bug report quality to improve bug localization with a two-phase model focusing on high-quality bug reports.

IRBL-related studies. Closely related to our work, Le et al. [22, 23] have proposed a study where they attempt to predict whether the ranked list produced by a bug localization tool is likely to be relevant to the given bug. They extract various textual and metadata features from 3 old projects and test on two IRBL techniques. They indeed find that it is possible, to some extent, to predict the effectiveness of the considered techniques. Our work is a generalized and large-scale investigation into the question of IRBL performance.

Wang et al. [43] have conducted an analytical study and a user study on IRBL techniques to assess their usefulness. Focusing on a single technique, BugLocator, and four common projects from previous studies, they report that the information needed for IR-based techniques to be effective is often not available in bug reports.

7 CONCLUSION

We presented a comprehensive reproducibility analysis of state-of-the-art IR-based bug localization techniques as a contribution to the community towards (1) demonstrating the actual performance of current approaches, (2) providing an updated benchmark for furthering this research field, and (3) showing the performance variations of different evaluation strategies.

Our reproducibility study have yielded several findings for the practice and research around bug localization. Overall, while IRBL approaches exhibit similar performance scores, subjects are not all equivalently adapted for each technique. All techniques are not overfitted to outdated subjects and perform better for up-to-date subjects as well. In contrast to the common belief, including test cases does not degrade the performance and, rather, even improves localization results. In addition, we reveal a potential research direction that leveraging duplicate reports together would enhance the performance of IRBL techniques.

Our future work will include (1) investigating relationships between project/report/file characteristics and the performance of different IRBL techniques (cf. D&C approach [19]), (2) building a decision model that predicts which IRBL technique performs better than others for a given project of file, and (3) improving preprocessing steps of IRBL techniques to reduce noise.

We provide a **replication package** with datasets and scripts as Bench4BL, at https://github.com/exatoa/Bench4BL

ACKNOWLEDGEMENTS

This work was supported by the Fonds National de la Recherche (FNR), Luxembourg, under projects RECOMMEND C15/IS/10449467, FIXPATTERN C15/IS/9964569. This work was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science, ICT & Future Planning) (No. 2015R1C1A1A01054994).

REFERENCES

[1] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund. 2007. On the Accuracy of Spectrum-based Fault Localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007.* 89 –98.

[2] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2006. An Evaluation of Similarity Coefficients for Software Fault Localization. In *12th Pacific Rim International Symposium on Dependable Computing, 2006. PRDC '06*. IEEE, 39–46.

[3] N. Bettenburg, R. Premraj, T. Zimmermann, and Sunghun Kim. 2008. Duplicate bug reports considered harmful . . . really?. In *IEEE International Conference on Software Maintenance, ICSM 2008.* IEEE, 337–345.

[4] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon. 2013. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE).* 188–197.

[5] Tegawendé F Bissyandé, Laurent Réveillère, Julia L Lawall, and Gilles Muller. 2012. Diagnosys: automatic generation of a debugging interface to the linux kernel. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on.* IEEE, 60–69.

[6] Tegawendé F Bissyandé, Ferdian Thung, Shaowei Wang, David Lo, Lingxiao Jiang, and Laurent Reveillere. 2013. Empirical evaluation of bug linking. In *17th European Conference on Software Maintenance and Reengineering (CSMR).* IEEE, 89–98.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research* 3 (March 2003), 993–1022.

[8] Brendan Cleary, Chris Exton, Jim Buckley, and Michael English. 2009. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering* 14, 1 (01 Feb 2009), 93–130.

[9] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6 (Sept. 1990), 391–407.

[10] Nicholas DiGiuseppe and James A. Jones. 2014. Fault density, fault types, and spectra-based fault localization. *Empirical Software Engineering* 20, 4 (March 2014), 928–967.

[11] B. Dit, A. Holtzhauer, D. Poshyvanyk, and H. Kagdi. 2013. A dataset from change history to support evaluation of software maintenance tasks. In *2013 10th Working Conference on Mining Software Repositories (MSR).* 131–134.

[12] B. Dit, E. Moritz, M. Linares-Vásquez, and D. Poshyvanyk. 2013. Supporting and Accelerating Reproducible Research in Software Maintenance Using TraceLab Component Library. In *2013 IEEE International Conference on Software Maintenance.* 330–339.

[13] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process* 25, 1 (Jan. 2013), 53–95.

[14] Bogdan Dit, Meghan Revelle, and Denys Poshyvanyk. 2013. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software. *Empirical Software Engineering* 18, 2 (April 2013), 277–309.

[15] William B. Frakes and Ricardo Baeza-Yates. 1992. *Information Retrieval: Data Structures and Algorithms* (1 ed.). Prentice Hall.

[16] G. Gay, S. Haiduc, A. Marcus, and T. Menzies. 2009. On the use of relevance feedback in IR-based concept location. In *2009 IEEE International Conference on Software Maintenance.* 351–360.

[17] N. Jalbert and W. Weimer. 2008. Automated duplicate detection for bug tracking systems. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, DSN 2008.* IEEE, 52–61.

[18] Dongsun Kim, Yida Tao, Sunghun Kim, and A. Zeller. 2013. Where Should We Fix This Bug? A Two-Phase Recommendation Model. *IEEE Transactions on Software Engineering* 39, 11 (Nov. 2013), 1597–1610.

[19] Anil Koyuncu, Tegawendé F. Bissyandé, , Kui Liu, Dongsun Kim, Jacques Klein, Yves Le Traon, and Martin Monperrus. 2018. *D&C: A Divide-and-Conquer, IR-based, Multi-Classifier Approach to Bug Localization.* Technical Report TR-2018-SerVAL-01. University of Luxembourg.

[20] Anil Koyuncu, Tegawendé F. Bissyandé, Dongsun Kim, Jacques Klein, Martin Monperrus, and Yves Le Traon. 2017. Impact of Tool Support in Patch Construction. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017).* ACM, 237–248.

[21] T. D. B. Le, M. Linares-Vásquez, D. Lo, and D. Poshyvanyk. 2015. RCLinker: Automated Linking of Issue Reports and Commits Leveraging Rich Contextual Information. In *2015 IEEE 23rd International Conference on Program Comprehension.* 36–47.

[22] Tien-Duy B. Le, Ferdian Thung, and David Lo. 2014. Predicting effectiveness of ir-based bug localization techniques. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on.* IEEE, 335–345.

[23] Tien-Duy B. Le, Ferdian Thung, and David Lo. 2016. Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. *Empirical Software Engineering* (2016), 1–43.

[24] Xiaoyong Liu and W. Bruce Croft. 2004. Cluster-based Retrieval Using Language Models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '04).* ACM, New York, NY, USA, 186–193.

[25] Stacy K. Lukins, Nicholas A. Kraft, and Letha H. Etzkorn. 2010. Bug localization using latent Dirichlet allocation. *Information and Software Technology* 52, 9 (Sept. 2010), 972–990.

[26] H. B. Mann. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (March 1947), 50–60.

[27] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval* (1 edition ed.). Cambridge University Press, New York.

[28] Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing* (1 edition ed.). The MIT Press, Cambridge, Mass.

[29] L. Moreno, J. J. Treadway, A. Marcus, and Wuwei Shen. 2014. On the Use of Stack Traces to Improve Text Retrieval-Based Bug Localization. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME).* 151–160.

[30] Anh Tuan Nguyen, Tung Thanh Nguyen, J. Al-Kofahi, Hung Viet Nguyen, and T. N. Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 263–272.

[31] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2012. Multi-layered approach for recovering links between bug reports and fixes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering.* ACM, New York, NY, USA, 63:1–63:11.

[32] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. 2016. Parameterizing and Assembling IR-Based Solutions for SE Tasks Using Genetic Algorithms. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 314–325.

[33] Chris Parnin and Alessandro Orso. 2011. Are automated debugging techniques actually helping programmers?. In *Proceedings of the International Symposium on Software Testing and Analysis.* 199–209.

[34] D. Poshyvanyk, Y. G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Transactions on Software Engineering* 33, 6 (June 2007), 420–432.

[35] Shivani Rao and Avinash Kak. 2011. Retrieval from Software Libraries for Bug Localization: A Comparative Study of Generic and Composite Text Models. In *Proceedings of the 8th Working Conference on Mining Software Repositories.* ACM, New York, NY, USA, 43–52.

[36] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of Duplicate Defect Reports Using Natural Language Processing. In *Proceedings of the 29th International Conference on Software Engineering, ICSE 2007 (ICSE '07).* IEEE Computer Society, Washington, DC, USA, 499–510.

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

[38] Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA.

[39] T Strohman, D Metzler, H Turtle, and WB Croft. 2004. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligence Analysis.* 2–6.

[40] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010.* ACM, Cape Town, South Africa, 45–54. ACM ID: 1806811.

[41] A. Sureka and P. Jalote. 2010. Detecting Duplicate Bug Report Using Character N-Gram-Based Features. In *17th Asia Pacific Software Engineering Conference, APSEC 2010.* IEEE, 366–374.

[42] G. Tassey. 2002. *The Economic Impacts of Inadequate Infrastructure for Software Testing: Final Report.* Diane Publishing Company. https://books.google.lu/books?id=juSgPAAACAAJ

[43] Qianqian Wang, Chris Parnin, and Alessandro Orso. 2015. Evaluating the usefulness of IR-based fault localization techniques. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis.* ACM, 1–11.

[44] Shaowei Wang and David Lo. 2014. Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization. In *Proceedings of the 22Nd International Conference on Program Comprehension.* ACM, New York, NY, USA, 53–63.

[45] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. 2016. Locus: locating bugs from software changes. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* ACM, Singapore, Singapore, 262–273.

[46] C. P. Wong, Y. Xiong, H. Zhang, D. Hao, L. Zhang, and H. Mei. 2014. Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis. In *2014 IEEE International Conference on Software Maintenance and Evolution.*

181–190.

[47] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (Aug. 2016), 707–740.

[48] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. 2014. CrashLocator: Locating Crashing Faults Based on Crash Stacks. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, New York, NY, USA, 204–214.

[49] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. ReLink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, New York, NY, USA, 15–25.

[50] Min Xie and Bo Yang. 2003. A study of the effect of imperfect debugging on software development cost. *IEEE Transactions on Software Engineering* 29, 5 (2003), 471–473.

[51] X. Ye, R. Bunescu, and C. Liu. 2016. Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation. *IEEE Transactions on Software Engineering* 42, 4 (April 2016), 379–402.

[52] K. C. Youm, J. Ahn, J. Kim, and E. Lee. 2015. Bug Localization Based on Code Change Histories and Bug Reports. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*. 190–197.

[53] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, Piscataway, NJ, USA, 14–24.