Empirical Software Engineering (2022) 27:47
https://doi.org/10.1007/s10664-021-10082-6

An empirical study of the effectiveness of IR-based bug localization for large-scale industrial projects

Wei Li[1] · **Qingan Li**[1] · **Yunlong Ming**[2] · **Weijiao Dai**[2] · **Shi Ying**[1] · **Mengting Yuan**[1]

Accepted: 10 November 2021Published online: 28 January 2022

Abstract
Bug localization, which refers to finding buggy files for a given bug report, is tedious and time-consuming for practical projects with tens of millions of lines of code. Recently, many information retrieval (IR)-based bug localization (IRBL) approaches have been proposed to formulate this problem as a search problem. Despite the excellent performance claimed in the literature, there is hardly any approach adopted in the industrial community to the best of our knowledge. The challenge of adapting IRBL to industrial projects is that the projects have different characteristics compared to open-source projects used in the literatures, which have not been taken into consideration in previous studies. In this paper, we re-implement six state-of-the-art IRBL techniques and evaluate their effectiveness on 10 Huawei projects consisting of 161,967 source code files and 24,437 bug reports in total. Localizing bugs in these projects faces several challenges, including the software product line, the bilingual issue, and the quality of bug reports, etc. We conduct comprehensive experiments to reveal how these factors affect IRBL effectiveness, and modify the data set to test whether some factors could be overcome, if additional information or hints are given. Based on the insights found in our work, we suggest potential improvements on IRBL techniques. This study is also expected to provide empirical evidences for other software tasks which face the same fundamental challenges.

Keywords Bug localization · Empirical study · Industrial projects

Communicated by: Aldeida Aleti, Annibale Panichella, and Shin Yoo

This article belongs to the Topical Collection: *Advances in Search-Based Software Engineering (SSBSE)*

The authors have no relevant financial or non-financial interests to disclose.

✉ Qingan Li
qingan@whu.edu.cn

✉ Mengting Yuan
ymt@whu.edu.com

Extended author information available on the last page of the article.

1 Introduction

Modern software projects consist of a large scale of source code files, with many developers getting involved. When a bug report is issued, it is hard for developers to find corresponding buggy files. In order to reduce the manual effort and promote the efficiency in debugging, various bug localization techniques have been proposed to automate the process of finding buggy files for given bug reports (Wong et al. 2016; Lee et al. 2018). However, as bug localization involves both natural languages and programming languages, it is still a challenging research topic due to the huge semantic gap between natural languages and programming languages.

A bug localization problem can be treated as a classification problem or a ranking problem. In the former, historical bug reports and source file revisions are used to train a classification function C. For a given bug report r and a source file s, $C(r, s)$ indicates whether s is correspondent to r. Some neural-network based approaches (Xuan et al. 2016; Xuan and Li 2017) have been proposed, aiming at training effective classifiers. However, the goal of the training process is to minimize a loss function, which is not obligated to quantify the relevance between bug reports and source files. In addition, training the classification suffers from the scalability problem, as its space complexity is in the order of the product of the number of bug reports and the number of source files. In the latter, information retrieval (IR) models are applied to rank the source code files based on their relevance to the bug report (Gay et al. 2009; Lukins et al. 2010; Nguyen et al. 2011; Zhou et al. 2012), and highly ranked files are supposed to be most likely to contain the buggy code. In this work, we focus on the latter approach, as it is well aligned to large-scale projects and suitable for the recommendation scenario.

Despite that IR-based bug localization (IRBL) has drawn great interest and attention from the academia, with numerous approaches claiming excellent performance in the literature, there is hardly any approach adopted in the developer community yet (Lee et al. 2018). Actually, the benchmarks used in research are much different from industrial projects. Most recent researchers conduct their experiments on the data set firstly proposed in Zhou et al. (2012), thus overfitting scenarios potentially exist. Although Lee et al. (2018) apply the existing IRBL techniques to dozens of new projects and obtain a comparable performance, most projects are composed of less than 1000 files, which is much smaller than industrial projects. Moreover, all the studies only explore the effectiveness on the open-source projects, whereas the large-scale industrial projects contain unique characteristics that are unsuitable for IRBL techniques.

In order to advance the application of IRBL techniques in real-world scenarios, we collect 10 closed-source Huawei projects and analyze the challenges faced in leveraging IRBL techniques. More than 160,000 source code files and 24,000 bug reports are involved in total. Compared to the open-source projects, these industrial projects in the data set have the following characteristics.

1) *Software product line (SPL):* Software product line (Paul and Linda 2002) is a common solution in project development of enterprises, where multiple products are manufactured from a single project by composing different features crafted in the project. Existing IRBL techniques are often misled by the wrong clues caused by the feature composition.
2) *Mixture of multiple natural languages:* Multilingualism is common in non-US projects, where the bug reports and code annotations tend to accommodate the native language and English. And it is difficult to establish semantic relations between sentences or phases written in different languages.
3) *Quality of bug reports:* We find that bug reports in industrial projects are of poorer quality than those in open-source projects, which is mainly caused by the following two

factors: (a) Multiple perspectives of bug reports—bug reports in industrial projects are committed by different stakeholders (e.g. end users, beta testers, etc) from different perspectives, which usually lack rich structured information (e.g. stack traces and program entity names) that hint at the buggy file. (b) Noises in bug reports—noisy bug reports frequently occur in real-world projects, most of which concern other software activities, such as optimization and refactoring activities, instead of corrective maintenance.

Furthermore, we re-implement six state-of-the-art IRBL techniques (Zhou et al. 2012; Wen et al. 2016; Wong et al. 2014; Saha et al. 2013; Wang and Lo 2014; Ye et al. 2014) and perform a comprehensive evaluation of these techniques on the collected Huawei projects by comparing with the performance on the open-source projects used in the literature. We also investigate whether the identified issues could be overcome by modifying the data set or providing additional hints. We report the following insights:

1) Despite the multiple industrial issues, IRBL techniques still work well on small-scale projects. However, the effectiveness on large-scale projects is comparatively lower and limited.
2) With only large-scale projects considered, the performance is stable across the projects, and the differences in performance among the state-of-the-art techniques become less apparent.
3) Many previous research studies (Zhou et al. 2012; Saha et al. 2013; Wong et al. 2014) report that the lexical similarity between bug reports and source code files is the major factor of IRBL. Due to the multiple perspectives and multilingualism problem, the lexical similarity is less effective in industrial projects. Instead, collaborative filtering mainly contributes to the effectiveness, thanks to the abundance of historical bug reports.
4) The performance of IRBL techniques on industrial projects can be further improved by eliminating noisy bug reports and leveraging SPL analysis.

Overall, the contributions of this paper are highlighted as follows.

- For the first time, we evaluate the effectiveness of IRBL techniques on large-scale, industrial projects. Six state-of-the-art techniques are re-implemented and the performance on open-source projects are used for comparison. During the evaluation, we find new insights that have never been discovered in open-source projects.
- We reveal major issues which prevent state-of-arts IRBL techniques from being applied to industrial projects. We also investigate the impacts of these issues on IRBL performance.
- With empirical evidences, we propose possible directions for improving current IRBL approaches. We also conduct preliminary experiments to validate parts of the directions.

As information retrieval techniques have been widely used in many other research fields in software engineering (e.g. code search Gu et al. 2018, code summarization Wan et al. 2018, traceability link recovery Mills and Haiduc 2017, etc.), and most of them face the same fundamental problems (e.g. the semantics gap problems, the SPL problem, etc), our empirical study could provide benefits for those researches and advances their applications in practical projects.

The remainder of this paper is organized as follows. Section 2 introduces the background of IRBL techniques and the motivation of this paper. Section 3 presents the design of our study. The analysis result is elaborated subsequently in Section 4. Section 5 discusses the potential improvements for IRBL techniques and Section 6 discusses the threats to validity. After surveying the related work in Section 7, we conclude our work in Section 8.

2 Background & Motivation

2.1 IR-Based Bug Localization

Information retrieval (IR) (Manning et al. 2010) is the activity of obtaining relevant information from a collection of resources (usually documents) that satisfies an information need (usually a query expressed as a set of key words). In the context of IR-based bug localization (IRBL), each bug report is construed as a query, while the set of source code files are considered as the document collection. The basic intuition is that the file most relevant to a bug report appears to be the most suspicious one that should be fixed to eliminate the bug.

Therefore, the primary task of IRBL technique is to design a model to measure the relevance between bug reports and source code files. A tendency observed in the recent work on IRBL is to combine more performance enhancement features to achieve a higher localization accuracy, where the features draw heavily on the software engineering domain knowledge. The following features have been mostly considered by state-of-the-art IRBL tools:

1) *Lexical similarity feature:* The lexical similarity between bug reports and source code files has been regarded as the crucial feature in IRBL techniques, owning to that bug reports usually share common words with their relevant defective code files. To compute the similarity, both the bug report and the source code are treated as plain text, and some text similarity models are applied, such as Vector Space Model (VSM) (Rao and Kak 2011), Probabilistic Topic Model (Lukins et al. 2010), and so forth. In addition, various refinements based on different heuristics have been proposed to further improve the effectiveness of this feature, some of which will be illustrated in Section 3.2.
2) *Collaborative filtering feature:* Collaborative filtering is a commonly used technique in recommender systems (Ekstrand et al. 2011), which refers to generating recommendations about the interests of a user by collecting preferences information from many other users. It has also been proven to be useful in bug localization given that similar bugs tend to fix similar files (Zhou et al. 2012). For a specific bug report, the collaborative filtering score of a source code file is calculated based on the textual similarities of their relevant historical bug reports and the number of files that are modified to fix each bug report.
3) *Stack-trace information feature:* Bug reports sometimes contain stack-trace information, which may provide direct clues to the buggy files (Wong et al. 2014). As revealed in the previous study by Schroter et al. (2010), the higher up the stack frames appear in stack-trace, the more likely it is to be bug-prone. Therefore, a separate stack-trace information score calculated based on the stack frames order is supposed to be helpful for improving the performance of bug localization.
4) *Version history feature:* It is suggested that the change history of source code files provides vital information for locating bugs. Generally the source files which were modified recently or frequently are more suspicious regarding the new-coming bugs (Kim et al. 2007; Ye et al. 2014). Based on this assumption, some features are inferred from the version history and integrated with other features.

2.2 Motivation

IRBL research has drawn great attention in academia and constantly produced new approaches that claim a higher performance over the state-of-the-art. However, some limitations lie in the performance assessments. Firstly, as demonstrated by Lee et al. (2018), most studies perform the evaluation on the same old and limited projects, which may lead to potential overfitting scenarios. Despite that Lee et al. explored an evaluation of existing techniques on dozens of new projects and achieved a comparable performance, most projects seems to be small-scale (less than 1000 source code files in a project). Akbar and Kak (2020) divided IRBL tools into three generations and evaluate these tools on some large-scale projects, but they focused on the comparison between different generations instead of effectiveness evaluation and the amount of bug reports they used is much small relative to the projects scales. Therefore a thorough evaluation on IRBL effectiveness for large-scale projects is still needed.

On the other hand, many existing approaches (Zhou et al. 2012; Saha et al. 2013; Wong et al. 2014; Wen et al. 2016; Saha et al. 2014; Lee et al. 2018) raise the threat to validity in their work that all the data sets used in research are collected from open-source projects, while the industrial data sets may have some different natures. Although Murali et al. (2020) explored the application of IRBL methods at a large industrial setting of Facebook, the bug reports in their setting are all automatically generated by crash or regression systems, which are completely different from the manually submitted ones. Hence, whether the existing IRBL techniques are generalizable to industrial projects still needs to be studied. Actually, we have found some specific industrial issues that may have adverse effects on the performance:

1) *Software product line:* Software product line (SPL) is a common solution in project development of enterprises, which refers to a set of products aggregate sharing a common, managed set of features and developed in a common project (Paul and Linda 2002). An SPL project usually involves some similar software features. For example, a network system may manage feature GPON and feature EPON simultaneously, each of which is a kind of optical communication technique. Source code distributed in similar features tends to be highly similar as well, which would interfere with the effectiveness of lexical similarity in IRBL techniques. In addition, similar bug reports may refer to bugs in different features, which would interfere with the effectiveness of collaborative filtering as well. For example, Fig. 1 provides two extremely similar bug reports in an SPL project. The first one reported a failure whose root cause was found in the

Bug Report 1
Summary: Boards failed.
Description: After activating the SPH patch and resetting the boards, some boards were not normal.
Modified feature: GPON Board.

Bug Report 2
Summary: Board failed to register.
Description: 1) Load the R47VSPH patch. 2) Reset 10G boards. The board was unable to register.
Modified feature: EPON Board.

Fig. 1 Two Similar bug reports in Huawei SPL projects (The summaries and descriptions are originally bilingual in English and non-English, which are translated into English only. So do the examples in Figs. 2 and 3)

code of feature “GPON Board”, while the root cause of the second one was located in the code of feature “EPON Board”. As a result, SPL brings challenges for IRBL techniques that rely heavily on similarity models. However, for most open-source projects with the SPL strategy, it is not an issue. The reason is that each feature is treated as an self-contained project, and the corresponding bug reports are committed to the feature project directly. For example, the project Eclipse has many features (e.g. PDE, JDT, etc.), each of which has its own bug report repository. In this case, previous research has not taken the SPL SPL issue into consideration. But for industrial SPL projects, this issue may lead to significant degradation of IRBL performance.

2) *Mixture of multiple natural languages:* In non-English speaking countries, developers tend to write code annotations in their native language for readability, while the program entities are still English words. Bug reports are usually bilingual as well. Considering that existing IRBL techniques (Zhou et al. 2012; Saha et al. 2013; Wang and Lo 2014) recommend potential buggy files mainly based on the lexical similarity between bug reports and source code files which ignores the semantic relations, the mixture of multiple natural languages will bring new challenges to IRBL research. There are multi-languages bug reports in open-source projects, and Xia et al. (2014) proposed CrosLocator which translates non-English textual descriptions of a bug reports into English and then applies a bug localization technique. But this approach is much ineffective when code files and bug reports both contain non-English text because if we translate all these text we would probably lose important information, and imprecise translation would make it more difficult to match the bug reports with related code files. The bilingual issue is also similar to the vocabulary mismatch problem in human-system communications (Furnas et al. 1987), where the search queries frequently mismatch a majority of the relevant documents. Query reformulation approaches (Zhao and Callan 2012; Rahman and Roy 2018b) are used to address this problem by expanding queries to improve the probability of finding relevant document. But all the existing methods are proposed in the monolingual settings and their effectiveness on bilingual issue still needs to be studied.

3) *Quality of bug reports:* The quality of bug reports in industrial projects is worse than that in open-source projects. According to the investigation of this study, we conclude the major factors of the quality issue as follows.

- *Multiple perspectives of bug reports:* Bug reports issued by submitters of different identities (e.g. end users, beta testers, etc.) are committed from different perspectives, as these stakeholders have different backgrounds in the aspect of knowledge related to the projects. For open-source projects, many bug reports are committed by experienced developers, who tend to describe bugs on source code level. Hence these bug reports generally contain rich structured information, such as stack traces and program entity names explicitly pointing to the bugs, which are proved to be useful in bug localization (Kim et al. 2011b; Zimmermann et al. 2010; Bettenburg et al. 2007). As the statistics gathered by Wang et al. (2015) from 10,000 SWT bug reports indicate: nearly one half contains the names of program entities. However, for industrial projects, most submitters are ignorant about the source code and describe bugs by describing the defective behaviors without providing code-related information, such as the example shown in Fig. 2. To make things worse, for many end users who barely have knowledge background of computers, they are still permitted to submit bug reports. Recent studies (Rahman and Roy 2018a; Wang et al.

A Bug Report from End User
Summary: Exception of management interface.
Description: Pictures on the management interface are displayed incompletely.

Fig. 2 A bug report submitted by an end user

2015) suggest that IRBL techniques do not perform well for the bug reports lacking rich structured information. Therefore, multiple perspectives of bug reports in industrial projects may impact the performance.

- *Noises in bug reports:* Earlier studies demonstrated that bug databases contain a large amount of noises (Bird et al. 2009; Herzig et al. 2013). Some bugs might be falsely linked with the modified source code files during the extraction process. And more are not really bugs, as they are concern with other software activities (e.g. perfective and adaptive maintenance, refactoring/restructuring activities, etc.), instead of corrective maintenance (Antoniol et al. 2008). For example, two noisy bug reports are given in Fig. 3. The first proposes the requirement of code optimization, and the second aims at dealing with the warnings raised by a static code analysis tool. At Huawei, most of these noisy bug reports are submitted by developers and are usually short and uninformative. There is little logical connection between the reports and the corresponding modified source files. In practical use of IRBL tools, we can locate bugs only for non-noisy bug reports. But in research, we can only collect the ground truth data from the historical bug reports, which did not have records about which type of issues the reports describe. Including noisy reports will negatively impact the performance evaluation for IRBL techniques.

In light of the limitations discussed above, we motivate this work to complement the performance evaluation of the current research and advance the practical application of IRBL techniques. For the first time, we explore an empirical evaluation of IRBL techniques on large-scale, industrial projects and investigate the impacts of the industrial issues on the performance of these techniques.

Noisy Bug Report 1
Summary: Code optimization for UDAP.
Description: Optimize the code for BRA and status inquiry in VAM module by reducing cyclomatic complexity.

Noisy Bug Report 2
Summary: fix lint warnings
Description: Several warnings still exist when inspecting the P300 library, needing to be fixed.

Fig. 3 Two noisy bug reports in Huawei projects

3 Study Design

3.1 Research Questions

Our empirical study focuses on the following research questions:

RQ1: *How effective are IRBL techniques for industrial projects?*

To answer this research question, we apply the IRBL techniques on the collected Huawei projects, and assess the effectiveness by comparing with that on commonly-used open-source projects.

RQ2: *How do industrial issues affect the effectiveness of IRBL techniques on industrial projects?*

We point out several industrial issues that prevent IRBL techniques from being well adapted to industrial projects. In this research question, we investigate and analyze the impacts of these issues on the effectiveness.

RQ3: *Can these issues be overcome?*

Based on the answer of RQ2, we study whether these industrial issues can be overcome. We conduct preliminary experiments to validate some of the ideas.

3.2 IRBL Techniques

A large number of IRBL techniques have been proposed in the literature. In this study, we consider six state-of-the-art, oft-cited techniques, as enumerated in Table 1. All these techniques, except LearningToRank, are re-implemented by Lee et al. (2018) in their reproducibility study. Hence, our experiments are well aligned to theirs for comparison.

1) *BugLocator* firstly combines lexical similarity feature and collaborative filtering feature. Additionally, it optimizes the classic VSM model by weighting the probability scores with file size, as findings show that larger files are more likely to contain bugs (Zhang 2009).
2) *BLUiR* optimizes lexical similarity feature by adopting structured information retrieval, where the code entities (class names, method names, variables names and comments) are extracted and matched with summaries and descriptions of bug reports separately.
3) *AmaLgam* integrates a version history feature into the BLUiR model to further improve the performance, based on the assumption that source files responsible for a recent bug are likely to still contain bugs.

Table 1 The studied IRBL techniques

Technique	Venue	Year
BugLocator (Zhou et al. 2012)	Intl. Conf. on Software Engineering	2012
BLUiR (Saha et al. 2013)	Intl. Conf. on Automated Software Engineering	2013
AmaLgam (Wang and Lo 2014)	Intl. Conf. on Program Comprehension	2014
BRTracer (Wong et al. 2014)	Intl. Conf. on Software Maintenance and Evolution	2014
LearningToRank (Ye et al. 2014)	Symp. on the Foundations of Software Engineering	2014
Locus (Wen et al. 2016)	Intl. Conf. on Automated Software Engineering	2016

4) *BRTracer* introduces the stack-trace feature. Also, it divides source code files into small segments to improve the effectiveness of lexical similarity feature given that bugs are generally localized in a small portion of the code.
5) *LearningToRank* extracts six features based on domain knowledge, including the lexical similarity, the collaborative filtering score, the bug-fixing recency score, the bug-fixing frequency score and so on. Then, a learning-to-rank approach is utilized to train the features' weights.
6) *Locus* leverages the software changes to enable more accurate and fine-grained bug localization. Specifically, it extracts the change logs and change hunks from the commit information and use them as an alternative of segments of source code files.

Note that Lee et al. (2018) also study the performance of another IRBL tool named BLIA (Youm et al. 2015), which can be perceived as AmaLgam integrated with the stack-trace information feature. However, almost no bug reports data set in our industrial data set contains stack-trace information, since the projects are all written in C and C++, both of which do not have native support for stack traces, and the released applications have all stripped the debug information. Therefore we do not consider this technique in this study. Instead, we investigate the effectiveness of LearningToRank due to its uniqueness in the way of determining features' weights. Besides, although Lee et al. (2018) have made their re-implementations available, only English-only Java projects are targeted. Considering that the industrial projects used in our study are all bilingual and written in C and C++, we re-implement these tools again and adapt them to our data set.

3.3 Evaluation Metrics

To evaluate the effectiveness of the existing IRBL techniques for industrial projects, we adopt the following three metrics which have been widely used in the literature.

1) *Top N Rank (Top@N):* This metric indicates the percentage of bug reports for which at least one corresponding buggy file is included in the top N (i.e. 1, 5, 10) results recommended by the IRBL tools. It is the most intuitive metric about which practitioners are most concerned, as a survey conducted by Kochhar et al. (2016) shows that most practitioners would accept the IRBL tool as effective when the position of the root cause is within the top 5.
2) *Mean Average Precision (MAP):* MAP considers whether all the corresponding buggy files tend to get highly ranked, thus it is favored in scenarios where users go deep in a ranked list to find more relevant results. It is calculated by taking the mean value of Average Precision (AP) scores for each bug report. The AP of a single bug report is defined as:

$$AP = \sum_{i=1}^{N} \frac{P(i) \cdot pos(i)}{\#\, of\ \ positive\ instances} \tag{1}$$

where N is the number of ranked files produced by IRBL tool, i is the rank. $pos(i)$ indicates whether the file at rank i is a buggy file. $P(i)$ is the precision at the given cut-off rank i and is calculated as:

$$P(i) = \frac{\#\, of\ \ positive\ instances\ in\ top\ i}{i} \tag{2}$$

3) *Mean Reciprocal Rank (MRR):* MRR emphasizes early precision and cares about the single highest-ranked relevant item. It is calculated by averaging the reciprocal rank

scores across all bug reports:

$$MRR = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{f\text{-}rank_i} \tag{3}$$

where M is the number of bug reports and $f\text{-}rank_i$ represents the position of first buggy file in the ranked list for the i-th bug report.

3.4 Study Objects

This study is intended to explore the effectiveness of the existing IRBL techniques for industrial, large-scale projects. For the sake of comparison, we also intentionally select two small-scale projects. Table 2 lists the Huawei projects used in our empirical evaluation, ordered by the number of source code files. 161,967 source code files and 24,437 bug reports are involved in total. All these projects are written in C and C++ and the bug reports are well-managed with Huawei internal Defect Tracking System.

The steps of collecting the dataset of Huawei projects are similar to those of collecting open-source dataset (Zhou et al. 2012). We have to establish the links between bug reports and related buggy source code files since no such information was recorded. First, we collect the bug reports of fixed bugs from Defect Tracking system. Then we scan through the change log of the source code repository (such as SVN and GIT) for the bug IDs in given formats (e.g. “fix bug DTS001”, etc.). The modified source code in commits for bug fixes is regarded as the related buggy file.

The first two projects are small-scale, in which less than 500 files are contained. And the remainders are much larger, which all suffer from the strenuous and time-consuming manual bug localization and are in urgent need of assistant tool to automate this process. Despite that most of these projects are system softwares of network equipments, they comprise low-level communication, network operating system, user interface, encryption and decryption, and so on, thus could cover a variety of scenarios of software application.

Table 2 also lists the number of noisy bug reports in each project. The noisy bug reports are detected by a simple heuristic rule, which will be introduced in Section 4.3. According to the statistics, a large proportion of bug reports are classified as noisy ones. The last column in Table 2 indicates the number of products involved in each project. Those projects that manage multiple products adopt the SPL strategy.

We also consider some open-source Java projects commonly used in the literature as listed in Table 3 for comparison. Note that some recent studies (Wen et al. 2016; Lee et al. 2018) used slightly different data sets from that collected by Zhou et al. (2012). Two sub-projects from Eclipse, namely PDE and JDT, substitute for Eclipse because the repository of Eclipse was separately managed for each sub-project afterwards. In this study, we focus on JDT and PDE instead of Eclipse as well.

For illustrating the impact of the industrial issues quantitatively, we present the comparison of vocabulary overlaps between bug reports and their related buggy files for industrial and open-source projects used in this study, as shown in Table 4. The second and third columns list the average number of unique terms after preprocessing in bug reports and source code files. Industrial bug reports are manifestly poorer in quality as much fewer terms are contained. The fourth to sixth columns are the percentages of bug reports that share different ranges of common terms with their related buggy files (For one bug report, We only consider the maximum number of terms shared with each buggy file, since some other files fixed for the bug may be incidentally modified and have little relationship with the bug).

Table 2 The studied industrial projects

Project	Description	Period	# Source files	# All BRs	# Noisy BRs	% Noisy BRs	# Products
NTA	A Network Traffic Analysis (NTA) system	02/15-12/18	118	57	1	1.8%	–
ESP	An Enterprise Simulation Platform (ESP)	06/16-07/18	482	1042	357	34.3%	–
DSLAM	A Digital Subscriber Line Access Multiplexer (DSLAM) system	01/15-03/19	6697	1825	712	39.0%	–
OLT	An Optical Line Terminal (OLT) system	01/15-05/19	6855	6109	1811	29.6%	–
IAS	An Integrated Access Software (IAS) system	01/15-04/19	14988	2128	993	46.7%	5
BSP	A Bottom Software Platform (BSP) that services for netBIOS	12/15-05/19	17462	586	112	19.1%	–
ONT	A system of Optical Network Terminal (ONT) equipments	01/15-06/18	26733	3219	912	28.3%	28
ANP	A series of Access Network Products (ANP)	07/15-08/18	27285	2757	638	23.1%	26
UTS	A Universal Time-sharing System (UTS)	01/15-05/19	29625	2266	214	9.4%	–
WDM	A system of Wavelength-Division Multiplexing (WDM) equipments	08/16-05/19	31722	4448	1263	28.4%	12

Table 3 The open-source Java projects

Project	Description	Period	# Source files	# Bug reports
ZXing	A barcode image processing Android library	03/10-09/10	391	20
SWT	An open source widget toolkit for Java	10/04-04/10	484	98
PDE	Plug-in Development Environment for Eclipse	08/10-04/15	5,273	60
AspectJ	An aspect-oriented extension to Java	12/02-05/10	6,485	286
JDT	A suite of Java development tools for Eclipse	07/14-04/15	6,842	94

Around 10 to 15% bug reports in most industrial projects do not share any terms with their buggy file, and only a little portion of bug reports share more than 10 terms. On average, about 4.5 to 6.5 terms are shared between the industrial bug reports and their buggy files as shown in the last column, while the number for open-source projects is twice or even three times as much. In summary, the lexical similarities between bug reports and buggy files in industrial projects are much lower than those in open-source projects. The industrial issues account for this, as poor-quality bug reports contain very few code-related terms and source code that are not well annotated share few non-English terms with bug reports.

3.5 Model Adaptation

Due to some different characteristics of industrial projects, we make the following adaptations when applying the existing IRBL techniques.

Table 4 Vocabulary overlap between bug reports and related buggy files

Projects	Avg. size of	Avg. size of	BRs sharing up to {} with one buggy file			Avg. num of
	bug reports	code files	0 terms	1–10 terms	$>$ 10 terms	shared terms
Industrial projects						
NTA	33.5	87.0	5.3%	71.4%	23.2%	7.3
ESP	26.4	69.3	6.9%	75.0%	18.0%	6.7
DSLAM	16.9	72.6	12.1%	79.5%	8.3%	4.9
OLT	17.8	74.8	9.0%	80.9%	10.1%	5.4
IAS	15.1	63.3	18.3%	75.0%	6.7%	4.5
BSP	30.2	45.9	13.7%	72.0%	14.3%	6.6
ONT	21.5	51.5	10.9%	77.6%	11.5%	5.9
ANP	23.4	53.1	12.1%	79.9%	8.0%	6.1
UTS	19.2	63.1	14.6%	80.5%	4.9%	4.3
WDM	21.6	41.0	8.3%	85.2%	6.4%	5.4
Open-source projects						
ZXing	72.7	57.3	0%	60%	40.0%	12.3
SWT	38.9	90.9	0%	51.0%	49.0%	12.2
PDE	45.4	59.4	1.7%	50.0%	48.3%	11.2
AspectJ	63.4	42.6	0%	28.3%	71.7%	16.9
JDT	73.2	59.3	1.1%	27.6%	71.3%	17.5

1) *Adaptation for Bilingualism:* The text similarity models used in the IRBL approaches generally require bug reports and source code to be represented by their constituent pre-processed tokens, thus tokenization is always performed for the text. For English, tokenization usually involves white space splitting, stopwords removal, stemming, etc. However, some non-English languages are written without spaces or other delimiters between the words, such as Chinese and Japanese. Tokenizing this kind of language needs an additional pre-processing step which is called word segmentation. In this study, all the industrial projects are bilingual. Specifically, the contents of bug reports are almost entirely Chinese (more than 80% Chinese tokens), with few words related to the product or module information in English. The code annotations are the same. We segment these sentences using Stanford Word Segmenter[1]—a CRF-based (Chang et al. 2008) segmentation tool developed by Stanford NLP Group.
2) *Adaptation for C/C++ projects:* Some IRBL techniques utilize the code construct information to improve effectiveness, such as BLUiR and AmaLgam. However, the available implementations of these tools are only applicable to Java projects. Given that all the industrial projects used in our study are written in C and C++, we adopt Eclipse C/C++ Development Tools (CDT) to build the abstract syntax tree (AST) of each source code file and extract the required program constructs as stated (Saha et al. 2013), including class names, method names, variable names and comments. Specially, since C is not object-oriented and has no class construct, we consider C file names as the substitution of class names, like Saha et al. (2014) do in their study.

4 Results & Analysis

Our experiments investigate and answer three research questions discussed in Section 3.1.

4.1 RQ1: Performance on Industrial Projects

To evaluate how effective IRBL techniques are for the industrial projects, we benchmark the performance against that on open-source Java projects. Note that all the studied techniques involve several parameters, which mainly are the weights of features and have a significant influence on the performance of bug localization. Except for LearningToRank, all the techniques tune the parameters using the whole dataset. For open-source projects, we directly adopt the optimal parameters published in the literature. For industrial projects, we retune these parameters to enable a better result because the different characteristics of projects may lead to different suitable parameters. Specifically, we focus on tuning the following parameters: (1) the weighting factor α for collaborative filtering feature score in BugLocator, BLUiR and BRTracer. (2) the weighting factor β for historical feature score in AmaLgam. Empirically, these approaches achieve the best performance on industrial projects when α varies between $0.5 \sim 0.7$ and β varies between $0.1 \sim 0.3$. For LearningToRank, the features' weights are trained using Ranking SVM (Joachims et al. 2017). We evaluate this tool with cross-validation, similar to that in Ye et al. (2014). The bug reports are first split into 10 folds. Then we repeatedly select different 9 folds of the bug reports for training the parameters, and validate the effectiveness on the remaining fold. All the results are aggregated lastly to form the final results.

[1] https://nlp.stanford.edu/software/segmenter.shtml

Tables 5 and 6 respectively show the results of studied IRBL techniques for open-source and industrial projects, in terms of Top@1, Top@5, Top@10, MAP and MRR. The last group in each table presents the aggregated results of each technique on all projects, which are calculated by assuming all bug reports are in a single project. The highest metric values for each project across the six techniques are highlighted in bold. Note that our results for

Table 5 The performance on open-source Java projects

Projects	IRBL tech.	Top@1	Top@5	Top@10	MAP	MRR
ZXing	BugLocator	0.2000	0.6500	0.7500	0.3306	0.3837
	BLUiR	0.5000	0.6500	**0.8000**	0.4996	0.5903
	AmaLgam	**0.5500**	0.6500	0.7500	**0.5234**	**0.6143**
	BRTracer	0.3000	0.5500	0.7500	0.3937	0.4219
	LearnToRank	0.2277	0.5025	0.6683	0.2331	0.3646
	Locus	0.4500	**0.7000**	**0.8000**	0.4609	0.5551
SWT	BugLocator	0.3571	0.6939	0.7959	0.4458	0.5016
	BLUiR	**0.5408**	0.7551	0.8367	**0.5684**	**0.6480**
	AmaLgam	0.5306	0.7449	0.8367	0.5613	0.6341
	BRTracer	0.4694	**0.7857**	**0.8776**	0.5257	0.5967
	LearnToRank	0.4592	0.7449	0.8367	0.5214	0.5988
	Locus	0.5000	**0.7857**	0.8571	0.5463	0.6213
PDE	BugLocator	0.3729	0.6440	0.6780	0.3955	0.4961
	BLUiR	0.3390	0.5085	0.5932	0.3607	0.4377
	AmaLgam	0.3390	0.5593	0.6101	0.3669	0.4458
	BRTracer	**0.4407**	**0.7119**	**0.8146**	0.4248	**0.5473**
	LearnToRank	0.2078	0.5260	0.5974	0.2551	0.3533
	Locus	0.4000	0.6667	0.7500	**0.4321**	0.5274
AspectJ	BugLocator	0.2762	0.5140	0.6538	0.2299	0.3989
	BLUiR	0.2657	0.4895	0.5979	0.2215	0.3782
	AmaLgam	0.2622	0.5140	0.6224	0.2194	0.3840
	BRTracer	**0.3392**	**0.5874**	**0.7098**	**0.2477**	**0.4455**
	LearnToRank	0.2422	0.5026	0.6536	0.2327	0.3726
	Locus	0.2553	0.4680	0.5319	0.2412	0.3541
JDT	BugLocator	0.2021	0.3830	0.4681	0.2011	0.2932
	BLUiR	0.2660	0.4787	0.5745	0.2643	0.3775
	AmaLgam	**0.2766**	0.5000	0.5745	0.2567	0.3843
	BRTracer	**0.2766**	0.4681	0.5745	0.2900	0.3832
	LearnToRank	0.2021	0.3723	0.4894	0.2008	0.2847
	Locus	**0.2766**	**0.5426**	**0.5851**	**0.3202**	**0.3876**
Aggregate	BugLocator	0.2856	0.5424	0.6535	0.2844	0.4090
	BLUiR	0.3303	0.5421	0.6426	0.3146	0.4395
	AmaLgam	0.3303	0.5619	0.6552	0.3125	0.4429
	BRTracer	**0.3610**	**0.6142**	**0.7292**	0.3279	**0.4717**
	LearnToRank	0.2693	0.5257	0.6526	0.2805	0.3952
	Locus	0.3244	0.5660	0.6310	**0.3365**	0.4325

Table 6 The performance on industrial projects (before eliminating noisy bug reports and without utilization of product information)

Projects	IRBL tech.	Top@1	Top@5	Top@10	MAP	MRR
NTA	BugLocator	0.3158	0.6667	0.8070	0.4403	0.4623
	BLUiR	0.3158	0.6140	0.7895	0.4368	0.4571
	AmaLgam	0.3158	0.6140	0.8070	0.4372	0.4574
	BRTracer	0.2982	0.6667	0.7895	0.4356	0.4525
	LearnToRank	**0.3509**	**0.7018**	**0.8421**	**0.4628**	**0.5137**
	Locus	**0.3509**	0.6316	0.8246	0.4606	0.4838
ESP	BugLocator	0.3781	0.6670	0.7668	0.4485	0.5049
	BLUiR	0.4031	0.6612	0.7735	0.4628	0.5203
	AmaLgam	0.4021	0.6747	0.7908	0.4704	0.5251
	BRTracer	0.3724	0.6651	0.7543	0.4429	0.4990
	LearnToRank	0.3426	**0.7140**	**0.8013**	0.4589	0.5086
	Locus	**0.5518**	0.7083	0.7793	**0.5927**	**0.6294**
DSLAM	BugLocator	**0.2197**	0.4005	0.4926	0.2224	0.3088
	BLUiR	0.2088	0.3934	0.4625	0.2109	0.3002
	AmaLgam	0.2153	0.4099	0.4816	0.2196	0.3116
	BRTracer	0.2175	**0.4230**	**0.5156**	**0.2263**	**0.3181**
	LearnToRank	0.1589	0.3551	0.4373	0.1843	0.2551
	Locus	0.1452	0.3145	0.3995	0.1702	0.2318
OLT	BugLocator	**0.2393**	**0.4456**	**0.5482**	**0.2454**	**0.3413**
	BLUiR	0.2272	0.4267	0.5292	0.2375	0.3284
	AmaLgam	0.2303	0.4382	0.5402	0.2412	0.3345
	BRTracer	0.2226	0.4416	0.5467	0.2351	0.3307
	LearnToRank	0.1946	0.4020	0.5074	0.2172	0.2979
	Locus	0.1822	0.3465	0.4421	0.2007	0.2673
IAS	BugLocator	0.1889	0.3496	0.4243	0.2309	0.2712
	BLUiR	0.1870	0.3731	0.4563	0.2342	0.2776
	AmaLgam	**0.2007**	**0.4008**	**0.4765**	**0.2515**	**0.2955**
	BRTracer	0.1875	0.3590	0.4431	0.2339	0.2742
	LearnToRank	0.1701	0.3346	0.4008	0.2137	0.2495
	Locus	0.1734	0.3228	0.4051	0.2143	0.2492
BSP	BugLocator	0.2253	0.4642	**0.5648**	0.2220	0.3367
	BLUiR	0.2218	0.4437	0.5512	0.2202	0.3289
	AmaLgam	0.2287	**0.4727**	0.5563	0.2377	0.3396
	BRTracer	0.2116	0.4505	0.5580	0.2180	0.3259
	LearnToRank	0.1621	0.4437	0.5410	0.2068	0.2856
	Locus	**0.2406**	0.4454	0.5307	**0.2641**	**0.3402**
ONT	BugLocator	**0.1960**	**0.4045**	0.4884	**0.2421**	**0.2954**
	BLUiR	0.1926	0.3979	0.4893	0.2379	0.2921
	AmaLgam	0.1935	0.4042	**0.4952**	0.2386	0.2930
	BRTracer	0.1945	0.4020	0.4930	0.2384	0.2932
	LearnToRank	0.1712	0.3703	0.4716	0.2247	0.2707
	Locus	0.1811	0.3647	0.4557	0.2286	0.2722

Table 6 (continued)

Projects	IRBL tech.	Top@1	Top@5	Top@10	MAP	MRR
ANP	BugLocator	0.1774	0.4229	0.5111	0.2147	0.2900
	BLUiR	**0.1806**	0.4222	0.5056	0.2141	0.2906
	AmaLgam	0.1788	0.4240	0.5074	0.2151	0.2896
	BRTracer	0.1785	0.4309	0.5281	0.2171	**0.2934**
	LearnToRank	0.1567	**0.4465**	**0.5303**	**0.2177**	0.2893
	Locus	0.1748	0.3544	0.4603	0.2056	0.2658
UTS	BugLocator	**0.1734**	**0.3822**	0.4687	**0.2006**	**0.2735**
	BLUiR	0.1721	0.3756	0.4590	0.1964	0.2700
	AmaLgam	0.1721	0.3738	0.4590	0.1974	0.2709
	BRTracer	**0.1734**	0.3791	**0.4704**	0.2000	0.2731
	LearnToRank	0.1249	0.3517	0.4373	0.1706	0.2276
	Locus	0.1377	0.2992	0.3839	0.1640	0.2192
WDM	BugLocator	0.1873	**0.4508**	**0.5603**	**0.2126**	**0.3105**
	BLUiR	0.1873	0.4296	0.5324	0.2027	0.3017
	AmaLgam	**0.1888**	0.4330	0.5360	0.2059	0.3048
	BRTracer	0.1835	0.4395	0.5461	0.2043	0.3042
	LearnToRank	0.1855	0.4200	0.5378	0.2119	0.2982
	Locus	0.1724	0.4060	0.5191	0.2025	0.2837
Aggregate	BugLocator	**0.2110**	**0.4314**	0.5263	**0.2370**	**0.3162**
	BLUiR	0.2077	0.4219	0.5156	0.2321	0.3110
	AmaLgam	0.2105	0.4311	0.5240	0.2369	0.3161
	BRTracer	0.2051	0.4307	**0.5287**	0.2328	0.3129
	LearnToRank	0.1802	0.4071	0.5039	0.2208	0.2887
	Locus	0.1893	0.3703	0.4656	0.2196	0.2798

open-source projects in Table 5 are slightly different from those in Lee et al. (2018). For BLUiR, AmaLgam and BRTracer, the reproducibility study (Lee et al. 2018) did not include the collaborative filtering feature in these approaches (though the original papers evaluated the effectiveness for both the cases of including and excluding this feature). Therefore our results are a little higher than theirs when we include this feature in these approaches. In addition, the results of BRTracer and BLUiR seem to be misplaced in Lee et al. (2018).

Firstly we assess the overall results by comparing the aggregated results shown in the two tables. It is obvious that the performance on open-source projects is superior over that on industrial projects. Specifically, for open-source projects, at most 36.1%, 61.4% and 72.9% bugs could be localized by inspecting top 1, 5 and 10 ranked files, while the percentages for industrial projects are only 21.1%, 43.1% and 52.9%. The highest aggregated MAP and MRR for open-source projects are 0.337 and 0.472 respectively, which are higher than the values in industrial projects (MAP = 0.237 and MRR = 0.326) by nearly 42% and 50%. We think that it is the industrial issues in the industrial projects that contribute to the low performance.

We then compare the performance across the projects. Note that the first two projects in both Table 5 (ZXing and SWT) and Table 6 (NTA and ESP) are much smaller than other projects, in which less than 1000 source code files are involved. We find that IRBL techniques perform well when applied to those small-scale projects, as more than 70% bugs and 80% of the bugs can be localized respectively within top 5 and top 10 ranked files. However, the performance on large-scale projects is relatively much lower, especially for industrial projects, where the Top@5 results range between 38% $\sim$ 47% and Top@10 results range between 47% $\sim$ 56% merely. The performance difference between small-scale and large-scale projects is understandable since locating bugs tend to be increasingly difficult with the project scaling up. Large projects are usually managed in more complicated structures and composed of more interactive components, while many tricky bugs might hide in the interaction between various components. In addition, We find that these IRBL techniques exhibit close performance in different industrial large-scale projects, where the optimal MAP varies between 0.201 $\sim$ 0.264 and the optimal MRR varies between 0.274 $\sim$ 0.341. But the performance variation in the open-source large-scale projects is significantly higher, given the achieved optimal MAP of 0.432, 0.248, 0.320 and the optimal MRR of 0.547, 0.446, 0.388 for PDE, AspectJ and JDT respectively. The ranges of MAP and MRR for open-source projects are as twice to three times as that for industrial projects. This might be caused by the different qualities of bug reports. The small quantity of bug reports involved in open-source projects might contribute to the instability of performance as well.

Furthermore, we compare the performance between different IRBL techniques. For open-source projects, BRTracer seems to be more effective according to the aggregated results shown in Table 5. BLUiR, AmaLgam and Locus are supposed to yield the similar performance. All the approaches outperform BugLocator except LearnToRank. The performance of LearnToRank is probably restricted by the limit number of bug reports, as training weights for features tend to require abundant data. For industrial projects, all the six techniques perform in line with each other, and none of them substantially outperforms the others. Unexpectedly, BugLocator is leading by a narrow margin when compared in terms of aggregated results. It seems that the improvement schemes of subsequent IRBL techniques are ineffective for the industrial projects. Actually, most of the improvements focus on optimizing the lexical similar feature, but this feature accounts for a tiny weight in the IR models for industrial projects in this study, which leads to the slight differences of performance among these techniques.

Finding 1. The state-of-the-art IRBL techniques tend to have an excellent performance when applied to small-scale projects. However, the performance degrades on large-scale projects. Considering the large-scale projects only, IRBL techniques yield lower performance on industrial projects.
Finding 2. The performance is stable across industrial large-scale projects, and the differences in performance among IRBL techniques is non-obvious.

4.2 RQ2: Impacts of Industrial Issues

We point out several kinds of industrial issues in this study, including the SPL issue, the multilingualism issue, the multiple perspectives of bug reports and the noises issue. In this section, we investigate the impacts of these issues on the effectiveness of IRBL techniques.

Impact of the Multiple Perspectives of Bug Reports and the Multilingualism Issue We discuss the impact of the two issues together for the following two reasons: (1) It is difficult to straightforwardly analyze their concrete impacts using some qualitative and contrast data analyses. (2) Both of the two issues mainly affect the lexical similarity feature in IRBL approaches by leading to a lower similarity score between bug reports and correspondent source files. In order to analyze their impacts, we explore the effectiveness of the lexical similarity feature.

Considering that all the studied IRBL techniques maximize their performance by combining multiple features as discussed in Section 2.1, we investigate the effectiveness of the lexical similarity feature by comparing with other features. Specifically, we focus on comparing the effectiveness between the lexical similarity feature and collaborative filtering feature as they are the primary ones. Other features are all proposed in later IRBL techniques and can be regarded as complementary parts.

To combine lexical similarity feature and collaborative filtering feature, a weighted sum function is generally applied in the studied techniques, which is defined as:

$$FinalScore = (1-\alpha) \times N(Score_L) + \alpha \times N(Score_C) \tag{4}$$

where $N(Score_L)$ and $N(Score_C)$ respectively indicates the normalized scores of lexical similarity feature and collaborative filtering feature, and α is used to adjust the weights between the two features. Thus, we can simply compare the effectiveness of features by comparing their corresponding weighting factors. As reported in the literature (Zhou et al. 2012), the performance reaches the best when α is between 0.2 and 0.3, which implies that the lexical similarity feature is more important in open-source projects.

We evaluate the impact of lexical similarity feature and collaborative filtering feature for industrial projects, by varying the value of α. This evaluation is conducted for all IRBL techniques that combine the two features, including BugLocator, BLUiR, AmaLgam and BRTracer. Since α has a consistent impact on each technique, we analyze the impact based on the results (in terms of MAP and MRR) of BugLocator in this paper, as shown in Fig. 4.

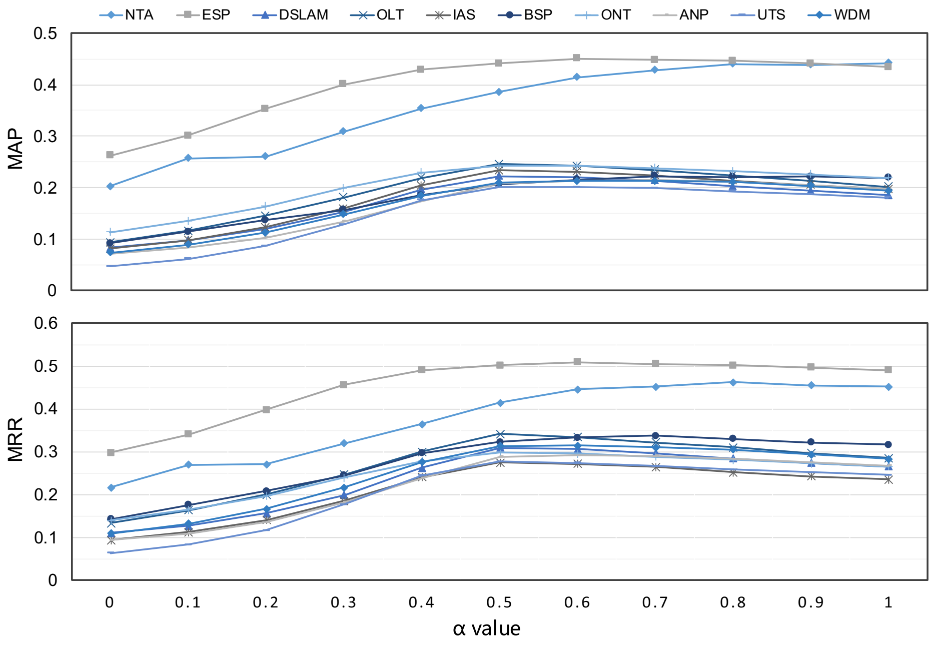

Fig. 4 Impact of varying α in BugLocator (α represents the weight of collaborative filtering feature)

We find that the performance is improved with the increase of α from 0 to 0.5. And these tools achieve the optimum performance when α is between 0.5 and 0.7. Therefore, the collaborative filtering feature brings a greater impact on the performance for industrial projects. We can also find that limited performance is achieved when considering lexical similarity feature only, as most the MAP and MRR values of industrial large-scale projects are below 0.1 and 0.15 respectively when α is set to 0.

As a result, the lexical similarity feature and collaborative filtering feature have inverse effectiveness for open-source projects and industrial projects. Combining the vocabulary overlap results shown in Table 4, we conjecture that due to the poor quality of bug reports and the multilingualism issue, the lexical similarity between bug reports and related buggy files tends to be small, which greatly impacts the effectiveness of bug localization. Remarkably, the experimental results indicate that the collaborative filtering feature exhibits higher effectiveness. Even using the collaborative filtering feature only (by setting $\alpha = 1$), we can get around 0.2 and 0.3 for MAP and MRR respectively. To clarify this, we manually analyze 300 random bugs that can be successfully located within the top 10 by this feature from all the projects. For each bug, the most 10 similar historical bug reports and their corresponding buggy files are extracted, with similarity information (e.g. lexical similarity values, common tokens between reports) computed for analysis. Among these 3000 similar bug report pairs, 845 (28.17%) pairs share at least one common buggy files. We suggest that (1) the abundant historical bug reports in industrial large-scale projects make it more possible to find much closer or even the same bugs (maybe incompletely resolved bugs appear again) whose relevant buggy files are responsible for the new bugs, and (2) some source code files are more vulnerable and more likely to contain bugs, and the collaborative filtering feature gives higher scores to these files since they have been modified for more times.

Impact of the SPL Issue To investigate the impact of the SPL issue, we divide the large-scale industrial projects into two groups: a group of non-SPL projects and a group of SPL projects. Note that the two small-scale projects NTA and ESP are excluded from this evaluation. We compare the difference of performance between the two groups. As all the IRBL approaches have similar performance, we take the case of BugLocator as the basis for discussion.

Table 7 shows the individual and aggregate performance values of non-SPL projects and SPL projects respectively. From the table, we can see that the aggregate results of SPL projects are slightly lower than those of non-SPL projects in terms of Top@5, Top@10, MAP and MRR. However, the difference between the Top@1 values is relatively significant, as the aggregate Top@1 value of SPL projects is about 15% lower than that of non-SPL projects. We conjecture that the low Top@1 result is related with the SPL issue because similar code in different software features tends to get close recommendation scores and the metric Top@1 is sensitive to the order of these files. The impacts to Top@5 and Top@10 are not so severe, as these metrics do not require the buggy files to rank first. Definitely, the low Top@1 result maybe related with some other factors, such as the number of source code files. In the next section, we investigate whether utilizing product information helps improve the performance, and the investigation will also reflect the impact of the SPL issue in a certain sense.

Impact of the Noises Issue After a comprehensive investigation of the defect tracking system and interviews with many developers/stakeholders in Huawei, we conclude that the noisy reports fall into the following categories: (1) Committed by routine maintenance tools like code style checker, program lint, etc. (2) Casually committed reports. Programmers

Table 7 The performance of BugLocator for non-SPL projects and SPL projects

Projects	Top@1	Top@5	Top@10	MAP	MRR
Non-SPL projects					
DSLAM	0.2197	0.4005	0.4926	0.2224	0.3088
OLT	0.2393	0.4456	0.5482	0.2454	0.3413
BSP	0.2253	0.4642	0.5648	0.2220	0.3367
UTS	0.1734	0.3822	0.4687	0.2006	0.2735
Aggregate	0.2214	0.4256	0.5230	0.2308	0.3213
SPL projects					
IAS	0.1889	0.3496	0.4243	0.2309	0.2712
ONT	0.1960	0.4045	0.4884	0.2421	0.2954
ANP	0.1774	0.4229	0.5111	0.2147	0.2900
WDM	0.1873	0.4508	0.5603	0.2126	0.3105
Aggregate	0.1876	0.4156	0.5080	0.2237	0.2955

may write casual reports for a code commitment, just to comply with company's regulations. (3) Generated by other software activities mentioned in the literature. For some part of these noises, we design heuristic rules to identify noisy report in the data set. Note that many noises are not easy to be recognized by using simple rules (e.g. most casually committed reports). As shown in Table 2, a large proportion of bug reports in the data set are classified as noises by our rules. For noisy reports, there is hardly any useful clue between the reports and the buggy files. IRBL algorithms are always misled by these noises.

Finding 3. For large-scale multilingual projects, the effectiveness of lexical similarity feature is severely restricted by the multiple perspectives of bug reports and the multilingualism problem. But collaborative filtering feature, which depends on the historical bug report information, plays a more important role due to the abundance of historical bug reports.

4.3 RQ3: Overcoming Industrial Issues

The multiple perspectives of bug reports and the multilingualism problem seem to be the primary causes that hamper the efficiency of IRBL techniques for industrial projects. However, overcoming the two issues requires some complex and innovative solutions, such as query reformulation (Rahman and Roy 2018b), cross-language information retrieval techniques (Oard and Diekema 1998), etc. These will be the new directions in our future work. In this study, we focus on investigating whether the noises issue and the SPL issue can be overcome when additional information or hints are provided.

1) *Eliminating Noises in Bug Reports*

To investigate whether eliminating noises helps improve the performance, we compare the performance before and after eliminating noises in bug reports. We propose a simple heuristic rule for identifying the noisy bug reports after analyzing a large amount of data and consulting with the relevant developers. We summarize a list of keywords, such as "cyclomatic complexity", "pc-lint", "fix warnings", etc. Bug reports that contain these keywords are regarded as noises, since they generally refer to other software activities rather than fixing a bug. Note that this rule is trivial but effective enough for our investigation.

As the noises have similar impact on each IRBL technique, we use BugLocator as an example for discussion and illustration. Table 8 shows the performance (in terms of MAP and MRR) of BugLocator in two different configurations: retaining and eliminating noisy bug reports. Mann-Whitney U test (Mann and Whitney 1947) is used to verify whether the differences are significant. For most projects, eliminating noisy bug reports helps improve both MAP and MRR values. But the extent of improvement varies with projects. For UTS and WDM, the MAP values are slightly increased by 0.0031 and 0.0018, and the MRR values are increased by 0.0011 and 0.0073. Comparatively, ESP and DSLAM have a decent performance improvement, where the MAP values are increased by 0.0445 and 0.037, and the MRR values are increased by 0.0413 and 0.0195.

Additionally, the performance of NTA and BSP decreases after removing noises in bug reports, but the decrease seems to be insignificant. The exceptions can be interpreted from the following two perspectives. (1) Inspecting noisy bug reports by key words is simplistic, which may lead to some misjudgments. Some noises might be omitted while some non-noisy bug reports might be misclassified. (2) Some noisy bugs are localizable, as their bug reports contain identifiable information that hints at buggy files. Moreover, for some noisy reports, there are a large number related files to fix. For example, sometimes refactoring an important class may modify most source files in a project. These noisy reports, however, are positive to IRBL performances.

2) *Utilizing Product Information*

As shown in Table 2, four of the projects adopt SPL strategy, namely IAS, ONT, ANP and WDM. These projects involve 5, 28, 26 and 12 products respectively. As the Huawei defect tracking system records a product information for each bug report, we verify whether the SPL problem can be alleviated by utilizing this information. In SPL projects, a product is composed of several software features, and tends to be independent from the other features. By giving the product information, IRBL is able to narrow the search range from the whole

Table 8 Comparison between performance results for BugLocator including and excluding noisy bug reports

Projects	Noisy bugs included		Noisy bugs excluded	
	MAP	MRR	MAP	MRR
NTA	0.4403	0.4623	↘ 0.4328	↘ 0.4617
ESP	0.4485	0.5049	↗ 0.4930**	↗ 0.5462**
DSLAM	0.2224	0.3088	↗ 0.2594**	↗ 0.3283*
OLT	0.2454	0.3413	↗ 0.2699**	↗ 0.3534**
IAS	0.2309	0.2712	↗ 0.2496	↗ 0.2830
BSP	0.2220	0.3367	↘ 0.2212	↘ 0.3317
ONT	0.2421	0.2954	↗ 0.2591*	↗ 0.3111*
ANP	0.2147	0.2900	↗ 0.2291*	↗ 0.3040*
UTS	0.2006	0.2735	↗ 0.2037	↗ 0.2746
WDM	0.2126	0.3105	↗ 0.2144	↗ 0.3178
Aggregate	0.2370	0.3162	↗ 0.2515**	↗ 0.3272**

$*$: p-value < 0.05, $**$: p-value < 0.01, ↗: increased, ↘: decreased repository down to the selected features. Note that there is not a particular product value for the core part shared by multiple products in these SPL projects. Therefore the core part will not be excluded for each product.

When providing product information, we also adapt the calculation method for collaborative filtering feature score and version history feature score by considering the historical bug reports with the same product only. This adaptation is based on the assumption that the fixed files of bugs with the same product are usually more responsible for the new coming bugs. Note that when calculating the lexical similarities in this setting, we use the same source code corpus and bug report corpus as that used in RQ1 instead of creating separate corpora for each product. This would have a limited effect on the results (we calculate the similarities based on tf-idf model, and only the idf value of words will be slightly affected).

Table 9 compares the performance of each IRBL techniques with and without utilization of product information for each SPL project (noisy bug reports are eliminated in this comparison). From the table, we can see that all the six techniques perform better when utilizing product information, except for Locus when applied to ONT. The improvement of performance for WDM is the most significant, as the performance of BugLocator improves from 0.214 to 0.248 MAP and from 0.318 to 0.363 MRR. However, the improvements for ONT are lower, given that the performance of BugLocator improves from 0.259 to 0.268 MAP and from 0.311 to 0.321 MRR. The different degrees of improvements might be induced by multiple complicated factors, such as the number of products, the structure of projects, the correctness of provided product information, etc.

The last group in Table 9 shows the aggregated results for each IRBL technique. It suggests that all the first five techniques have similar improvements in terms of MAP and MRR, while the improvements of Locus is less significant. This is mainly caused by the strategy of using product information. We adjust the procedure of calculating collaborative filtering feature score when considering product information, but Locus does not involve this feature.

Finding 4. Eliminating noisy bug reports and utilizing product information generally help improve the performance of bug localization. For further improvement, IRBL techniques can consider exploring sophisticated noise detection algorithms and advanced SPL analysis.

4.4 Feedback from Practitioners

Lastly, We surveyed 25 Huawei practitioners who are working on our evaluated projects and collected feedback about their willingness to adopt IRBL techniques and their satisfaction level with the performance of existing IRBL techniques.

As shown in Fig. 5, around two-thirds (68%) of the practitioners stated that they would be willing or very willing to use IRBL techniques when resolving bugs. Only 3 practitioners said they were generally unwilling, and the major reason is that they do not believe that IRBL tools can help locate difficult bugs, and they prefer traditional debugging ways by setting breakpoints or single-stepping which can help them understand program behavior. The others held a neutral attitude towards these techniques as they are not sure about their effectiveness.

To evaluate practitioners' satisfaction level with the performance of existing IRBL techniques, we surveyed on their expectation for Top@1 and Top@5 accuracy. The percentages of practitioners who were satisfied with different result values are shown in Fig. 6. Only a small proportion of practitioners (less than 20%) accepted the Top@1 result of 20% and

Table 9 Comparison between performance results with and without utilizing product information

Projects	IRBL tech.	Without utilization		With utilization	
		MAP	MRR	MAP	MRR
IAS	BugLocator	0.2496	0.2830	↗ 0.2736**	↗ 0.3099**
	BLUiR	0.2471	0.2798	↗ 0.2727**	↗ 0.3068**
	AmaLgam	0.2644	0.2984	↗ 0.2916**	↗ 0.3264**
	BRTracer	0.2511	0.2860	↗ 0.2722**	↗ 0.3139**
	LearnToRank	0.2287	0.2609	↗ 0.2452*	↗ 0.2796*
	Locus	0.2357	0.2670	↗ 0.2560**	↗ 0.2896**
ONT	BugLocator	0.2591	0.3111	↗ 0.2678*	↗ 0.3211
	BLUiR	0.2511	0.3042	↗ 0.2584	↗ 0.3125
	AmaLgam	0.2537	0.3066	↗ 0.2599	↗ 0.3142
	BRTracer	0.2562	0.3093	↗ 0.2695**	↗ 0.3238**
	LearnToRank	0.2439	0.2899	↗ 0.2482	↗ 0.2949
	Locus	0.2238	0.2650	↘ 0.2171	↘ 0.2618
ANP	BugLocator	0.2291	0.3040	↗ 0.2486**	↗ 0.3215*
	BLUiR	0.2243	0.3009	↗ 0.2476**	↗ 0.3263**
	AmaLgam	0.2251	0.2993	↗ 0.2490**	↗ 0.3216**
	BRTracer	0.2307	0.3054	↗ 0.2513**	↗ 0.3246*
	LearnToRank	0.2320	0.2985	↗ 0.2525**	↗ 0.3227**
	Locus	0.2141	0.2697	↗ 0.2230*	↗ 0.2789
WDM	BugLocator	0.2144	0.3178	↗ 0.2478**	↗ 0.3639**
	BLUiR	0.2017	0.3043	↗ 0.2353**	↗ 0.3519**
	AmaLgam	0.2054	0.3079	↗ 0.2388**	↗ 0.3544**
	BRTracer	0.2069	0.3120	↗ 0.2411**	↗ 0.3589**
	LearnToRank	0.2098	0.3012	↗ 0.2332**	↗ 0.3338**
	Locus	0.2035	0.2845	↗ 0.2153**	↗ 0.2974**
Aggregate	BugLocator	0.2343	0.3082	↗ 0.2566**	↗ 0.3353**
	BLUiR	0.2261	0.3003	↗ 0.2492**	↗ 0.3295**
	AmaLgam	0.2306	0.3042	↗ 0.2537**	↗ 0.3322**
	BRTracer	0.2314	0.3063	↗ 0.2557**	↗ 0.3355**
	LearnToRank	0.2266	0.2923	↗ 0.2434**	↗ 0.3138**
	Locus	0.2156	0.2735	↗ 0.2229**	↗ 0.2825*

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

the Top@5 result of 40%. To achieve a satisfaction rate of 80% among the practitioners, the Top@1 and Top@5 results need to reach nearly 60% and 80% respectively, which are significantly higher than the results of existing IRBL techniques. Practitioners generally have a high expectation because they think that inspecting wrongly recommended code files will aggravate their burden of locating bugs. Besides, a low localization accuracy also shakes their confidence in IRBL tool.

Briefly, most practitioners are willing to adopt IRBL tools in bug localization processes, but existing techniques have not met their desired extent. In order to apply IRBL techniques to practice, we still need to improve the performance substantially.

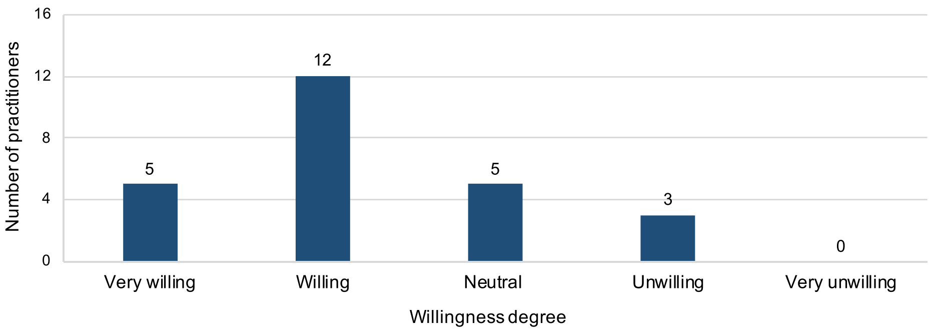

Fig. 5 Practitioners' willingness to adopt IRBL techniques

5 Potential Improvements for IRBL

Our experiment results suggest that there is still much room for improving IRBL techniques. Following are some insights for the improvements.

1) *Leveraging program analysis techniques:* Most existing IRBL approaches treat bug reports and source codes as plain text. Although BLUiR separates different parts of code (class names, method names, variables names and comments) to improve matching, fundamentally it processes programming languages as structured natural languages. Given more program information like control-flow graphs and data-flow information, it is possible to locate buggy file more precisely. Consider the bug report 1 in Fig. 1, although the key word "SPH" occurs in the report, the buggy file itself does not contain the key word. The reason is that the key word is stored in another file as a string resource and is used by the buggy file. Given some data-flow information, we are able to list all source files that use the "SPH" string as suspicious buggy files, when a report containing "SPH" is issued. For SPL projects, advanced program analysis techniques would help locate the root cause of a report in a complex product-feature-module hierarchy. Locating at a finer level enables a higher localization accuracy. For example, if we provide the feature information (analysed after the fact) for each bug report in project WDM, the performance of BugLocator would get a significant increase as shown in Table 10. Hence, leveraging sophisticated program analysis technique is able to boost the performance of IRBL in large-scale projects, as well as SPL projects.

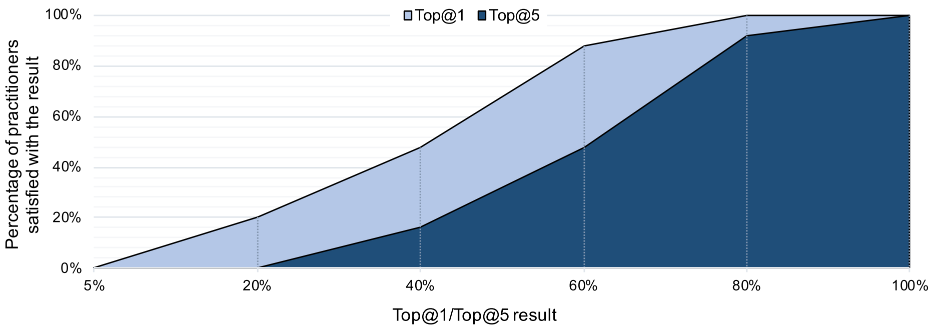

Fig. 6 Practitioners' expectation for Top@1 and Top@5 results

Table 10 The performance gain of BugLocator after providing feature information for WDM

	Top@1	Top@5	Top@10	MAP	MRR
Before	0.2361	0.5183	0.6144	0.2478	0.3639
After	0.3419	0.6025	0.7002	0.3225	0.4612

2) *Utilizing Cross Language Information Retrieval (CLIR) techniques:* CLIR (Oard and Diekema 1998) is a subfield of information retrieval, where queries and documents are expressed in different languages. Various CLIR techniques have been proposed and applied to both research and practice, and the most accurate techniques are supposed to be nearly as effective as monolingual information retrieval. Considering that existing IRBL techniques is less effective for multilingual projects due to the lexical gap problem, we can introduce some CLIR techniques to mitigate this problem.
3) *Improving effectiveness of collaborative filtering feature:* Our experiment results indicate that collaborative filtering feature can be as important as or even more important than lexical similarity feature. Thus the performance of bug localization can be further improved by optimizing collaborative filtering algorithm. We provide the following two possible improvement schemes: (1) Existing IRBL techniques aggregate similarities of all historical bug reports when calculating collaborative filtering scores for source code files. However, the effectiveness of this feature might be negatively affected when the historical bug reports are excess, as most historical bugs are extraneous to the new coming bugs and source code files that have been modified for many times tend to get a higher score. To alleviate this problem, we can enforce a threshold for similarity or the number of similar bug reports. (2) Similarities between bug reports are mainly calculated based on VSM model now, which ignores the latent semantic relationships. Ye et al. (2016) propose to calculate document similarities using word embeddings. However, this approach requires a large scale of corpus (such as API documents, tutorials, etc.) to train the word embeddings. Fortunately, some state-of-the-art deep language models (such as BERT (Devlin et al. 2018)), which have been pre-trained for generous purpose on enormous amount of unlabeled data, can be applied to specific tasks by simply fine-tuning. They can be applied to the similarity problem by converting it to the classic two-sentences task.
4) *Detecting noises in intelligent ways:* In this study, we find that eliminating noisy bug reports can improve the performance of bug localization. Actually, some studies have explored approaches for automatically detecting noises (Kim et al. 2011a; Pingclasai et al. 2013; Qin and Sun 2018). Text classification models and clustering models are respectively applied to distinguish between clean bugs and noisy bugs in supervised and unsupervised approaches. These approaches have been shown to be helpful in the field of defect prediction. Thus we can also attempt to apply them to IR-based bug localization.

6 Threats to Validity

There are some potential threats to the validity of our study.

1) This study comprehensively evaluates the performance of IRBL techniques on large-scale, industrial projects. However, all the industrial projects come from Huawei and

may not represent projects from other companies. In the future, we will perform a larger scale evaluation on the projects from different companies.

2) This study selects five open-source Java projects commonly used in the literature for comparison. These projects are somewhat outdated and scanty, which may raise several threats to validity as the comparison may be biased and not comprehensive enough. Although a set of new projects are proposed in a recent reproducibility study (Lee et al. 2018), we did not use these new projects based on the following concerns: (1) most of these projects are much small (35 of 46 projects are composed of less than 1000 source files, while only 3 projects contain more than 5000 source files). (2) There were big discrepancies in IRBL performance among these projects (the highest MAP for each project ranges from 0.04 to 1.0 and MRR ranges from 0.05 to 1.0). Over high or over low results are apparently special cases, making it difficult to draw general conclusions for open-source projects. (3) The old projects are still the most representative and widely-used benchmarks in the literatures. And the aggregated MAP and MRR results of new open-source projects are close to those of old open-source projects. Considering that the aim of this study is to analyze the challenges of adapting IRBL to industrial projects, the old projects will not bias our analysis in comparison.
3) For each project, we use a single release for bug localization. Bug reports are removed if their correspondent buggy files are all excluded from this release. The single release strategy simplifies the procedure of evaluation and is adopted by most IRBL studies. For more accurate assessment, we can simulate the realistic situation by checking a before-fix version for each bug report. But it is not scalable for a large-scale experiment.
4) This study regards all the modified files for a bug fix commit as the buggy files, like existing studies on open-source projects. However, some files fixed for a bug may be incidentally modified and have little relationship with the bug. Including those files imposes a certain extent of threat to the validity of the study. Removing incorrect fixed files is much impractical for large-scale projects as (1) it takes much effort of practitioners to identify the incorrect files (source code and modifications need to be reviewed for each bug), and (2) some modified files which seem incorrect may not be directly related with the bugs but implicitly interact with the direct buggy files.

7 Related Work

We review briefly previous work related to the two aspects: IR-based bug localization and IRBL-related studies.

7.1 IR-Based Bug Localization

IR-based bug localization has been an active area of research for over two decades. Various IR models have been applied, such as Latent Semantic Indexing (LSI) (Poshyvanyk et al. 2007), Latent Dirichlet Allocation (LDA) (Lukins et al. 2010), and Vector Space Model (VSM) (Zhou et al. 2012). As VSM model shows better performance than others, it has been widely used in latter researches. Recent studies also advocate for combing multiple features gathered from the software to improve IRBL performance. These features include structured information (Saha et al. 2013; Wang and Lo 2014), historical bug reports (Zhou et al. 2012; Wang and Lo 2014), version history (Sisman and Kak 2012; Wang and Lo 2014; Wen et al. 2016), stack traces (Wong et al. 2014), and so on. Furthermore, some studies consider adopting machine learning techniques. Ye et al. (2014) proposed to optimize weights for different features using learning-to-rank technique. Lam et al. (2015, 2017) proposed to incorporate deep learning into IR-based bug localization to alleviate the vocabulary mismatch problem.

7.2 IRBL-Related Studies

Apart from proposing novel IRBL techniques, a large number of empirical studies are conducted on the area of IR-based bug localization. Saha et al. (2014) conducted a study investigating the effectiveness of an IRBL technique (specifically BLUiR) for C programs, and suggested that the performance is comparable with that on Java programs. Le et al. (2014, 2017) attempted to predict whether a recommendation for a given bug report is likely to be effective or not, supposing the recommendation to be effective if the top-N ranked files produced by IRBL tools include at least one buggy file. Wang et al. (2015) conducted an analytical study evaluating the usefulness of IRBL techniques. They pointed out that these techniques do not perform well if bug reports lack rich structured information.

Some researches focus on improving the quality of queries formulated from the full text of bug reports. Mills et al. (2017) introduced an automatic query quality prediction approach which can be used to identify low-quality bug reports whose responsible buggy files can hardly be located by IRBL techniques. The identified low-quality reports can be further reformulated manually or by other reformulation techniques. Mills et al. (2020) devised a Genetic Algorithm (GA) that selects words from a bug report vocabulary to construct a near-optimal query and provided the evidence on the potential improvements of IRBL effectiveness by optimizing queries. However, they only presented a preliminary study as their GA is based on the condition that the ground truth is known and cannot be used in a real bug localization scenario. Chaparro et al. (2019) proposed to reformulate queries by selecting existing information from bug reports (including title, observed behavior, expected behavior, code snippets, etc.) and remove the irrelevant parts. This approach requires the bug reports to contain rich structured information. Rahman and Roy (2018b) presented BLIZZARD, which first classifies bug reports according to whether there are excessive program entities in the description, and then applies appropriate reformulations to each category. For bug reports containing only unstructured natural language description, BLIZZARD complements them with appropriate keywords from top ranked source code files returned by the IRBL tool for the initial query. The effectiveness of these reformulation techniques on industrial large-scale projects will be explored in our future study.

Mostly closely related to our work, Lee et al. (2018) conducted a reproducibility study on the performance of IRBL techniques. They considered that the benchmarks used in existing researches have not yet reached the level of maturity, thus collected 46 new Java projects and performed a comprehensive evaluation. Their experiment results showed that the IRBL performance is higher on the new projects. However, most of these projects are small-scale and involve less than 1000 source code files. Akbar and Kak (2020) divided existing IRBL techniques into three generations, and performed a comparative evaluation on large-scale projects in multiple programming languages. They focused on the comparison between different generations and found that those IRBL techniques, which exploit proximity, order, and semantic relationships between the terms of bug reports and source code, have a better performance. However, their study did not thoroughly evaluate the IRBL effectiveness on large-scale projects and they only considered open-source projects, while industrial projects might possess some special characteristics that have an adverse impact on IRBL performance.

8 Conclusion

To advance the application of IR-based bug localization in real-world scenarios, we perform an empirical study on the effectiveness of six state-of-the-art IRBL techniques for Huawei large-scale projects. Through the empirical study, we identify some industrial issues existing in the industrial projects, including SPL problem, multilingualism problem and poor quality of bug reports. These issues lead to a degradation of performance compared with that on open-source projects. We further conduct some experiments revealing that the performance can be improved by minimizing the impact of some issues. However, some limitations exist as approaches to overcoming the multiple perspectives of bug reports and the multilingualism problem are not investigated in this study. Lastly, we propose some potential improvements on IRBL techniques in this paper.

To our best knowledge, this is the first work for evaluating the industrial practice related to IR-based bug localization. We believe that our study is helpful for researchers and practitioners to further improve current IRBL techniques. We also expect that our study could provide benefits for some other IR-related researches in software engineering.

Acknowledgements This work is partially supported by Natural Science Foundation of China (No. 61872272, 61640221), and the Strategic Priority Research Program of Chinese Academy of Science (No.XDC05040100).

References

Akbar SA, Kak AC (2020) A large-scale comparative evaluation of IR-based tools for bug localization. In: Proceedings of the 17th international conference on mining software repositories, pp 21–31

Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc Y-G (2008) Is it a bug or an enhancement?: a text-based approach to classify change requests. In: CASCON 8, pp 304–318

Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T (2007) Quality of bug reports in Eclipse. In: Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange, pp 21–25. https://doi.org/10.1145/1328279.1328284

Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P (2009) Fair and balanced?: bias in bug-fix datasets. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 121–130

Chang PC, Galley M, Manning CD (2008) Optimizing Chinese word segmentation for machine translation performance. In: Proceedings of the third workshop on statistical machine translation, pp 224–232

Chaparro O, Florez JM, Marcus A (2019) Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empir Softw Eng 24(5):2947–3007

Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805

Ekstrand MD, Riedl JT, Konstan JA (2011) Collaborative filtering recommender systems. Foundations and Trends®, in Human–Computer Interaction 4(2):81–173

Furnas GW, Landauer TK, Gomez LM, Dumais ST (1987) The vocabulary problem in human-system communication. Commun ACM 30(11):964–971. https://doi.org/10.1145/32206.32212

Gay G, Haiduc S, Marcus A, Menzies T (2009) On the use of relevance feedback in IR-based concept location. In: 2009 IEEE international conference on software maintenance, pp 351–360

Gu X, Zhang H, Kim S (2018) Deep code search. In: 2018 IEEE/ACM 40th international conference on software engineering (ICSE), pp 933–944

Herzig K, Just S, Zeller A (2013) It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 international conference on software engineering, pp 392–401

Joachims T, Swaminathan A, Schnabel T (2017) Unbiased learning-to-rank with biased feedback. In: Proceedings of the tenth ACM international conference on web search and data mining, pp 781–789

Kim S, Zimmermann T, Whitehead EJ Jr, Zeller A (2007) Predicting faults from cached history. In: Proceedings of the 29th international conference on Software Engineering, pp 489–498

Kim S, Zhang H, Wu R, Gong L (2011a) Dealing with noise in defect prediction. In: 2011 33rd international conference on software engineering (ICSE), pp 481–490

Kim D, Tao Y, Kim S, Zeller A (2011b) Where should we fix this bug? A two-phase recommendation model. IEEE Trans Softw Eng 39(11):1597–1610. https://doi.org/10.1109/TSE.2013.24

Kochhar PS, Xia X, Lo D, Li S (2016) Practitioners' expectations on automated fault localization. In: International symposium on software testing & analysis, pp 165–176

Lam AN, Nguyen AT, Nguyen HA, Nguyen TN (2015) Combining deep learning with information retrieval to localize buggy files for bug reports (n). In: 2015 30th IEEE/ACM international conference on automated software engineering (ASE), pp 476–481

Lam AN, Nguyen AT, Nguyen HA, Nguyen TN (2017) Bug localization with combination of deep learning and information retrieval. In: 2017 IEEE/ACM 25th international conference on program comprehension (ICPC), pp 218–229

Le T-DB, Thung F, Lo D (2014) Predicting effectiveness of IR-based bug localization techniques. In: 2014 IEEE 25th international symposium on software reliability engineering, pp 335–345

Le T-DB, Thung F, Lo D (2017) Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. Empir Softw Eng 22(4):1–43

Lee J, Kim D, Bissyandé TF, Jung W, Le Traon Y (2018) Bench4BL: reproducibility study on the performance of IR-based bug localization. In: Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis, pp 61–72

Lukins SK, Kraft NA, Etzkorn LH (2010) Bug localization using latent dirichlet allocation. Inf Softw Technol 52(9):972–990

Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 50–60

Manning C, Raghavan P, Schütze H (2010) Introduction to information retrieval. Nat Lang Eng 16(1):100–103

Mills C, Haiduc S (2017) The impact of retrieval direction on ir-based traceability link recovery. In: 2017 IEEE/ACM 39th international conference on software engineering: new ideas and emerging technologies results track (ICSE-NIER), pp 51–54

Mills C, Bavota G, Haiduc S, Oliveto R, Marcus A, De Lucia A (2017) Predicting query quality for applications of text retrieval to software engineering tasks. In: ACM Trans Softw Eng Methodol (TOSEM), vol 26, pp 1–45

Mills C, Parra E, Pantiuchina J, Bavota G, Haiduc S (2020) On the relationship between bug reports and queries for text retrieval-based bug localization. Empir Softw Eng 25(5):3086–3127

Murali V, Gross L, Qian R, Chandra S (2020) Industry-scale IR-based bug localization: a perspective from Facebook. arXiv:2010.09977

Nguyen AT, Nguyen TT, Al-Kofahi J, Nguyen HV, Nguyen TN (2011) A topic-based approach for narrowing the search space of buggy files from a bug report. In: Proceedings of the 2011 26th IEEE/ACM international conference on automated software engineering, pp 263–272

Oard DW, Diekema AR (1998) Cross-language information retrieval. Annu Rev Inf Sci Technol (ARIST) 33:223–56

Paul C, Linda N (2002) Software product lines: patterns and practice. Addison-Wesley, Boston

Poshyvanyk D, Gueheneuc Y-G, Marcus A, Antoniol G, Rajlich V (2007) Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Trans Softw Eng 33(6):420–432

Pingclasai N, Hata H, Matsumoto K-I (2013) Classifying bug reports to bugs and other requests using topic modeling. In: 2013 20th Asia-Pacific software engineering conference (APSEC) 2, pp 13–18

Qin H, Sun X (2018) Classifying bug reports into bugs and non-bugs using LSTM. In: Proceedings of the tenth asia-pacific symposium on Internetware, p 20

Rahman MM, Roy C (2018a) Poster: improving bug localization with report quality dynamics and query reformulation. In: 2018 IEEE/ACM 40th international conference on software engineering: companion (ICSE-companion), pp 348–349

Rahman MM, Roy CK (2018b) Improving ir-based bug localization with context-aware query reformulation. In: Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering, pp 621–632

Rao S, Kak A (2011) Retrieval from software libraries for bug localization: a comparative study of generic and composite text model. In: Proceedings of the 8th working conference on mining software repositories, pp 43–52

Saha RK, Lease M, Khurshid S, Perry DE (2013) Improving bug localization using structured information retrieval. In: 2013 28th IEEE/ACM international conference on automated software engineering (ASE), pp 345–355

Saha RK, Lawall J, Khurshid S, Perry DE (2014) On the effectiveness of information retrieval based bug localization for c programs. In: 2014 IEEE international conference on software maintenance and evolution, pp 161–170

Schroter A, Schröter A, Bettenburg N, Premraj R (2010) Do stack traces help developers fix bugs? In: 2010 7th IEEE working conference on mining software repositories (MSR 2010), pp 118–121

Sisman B, Kak AC (2012) Incorporating version histories in information retrieval based bug localization. In: 2012 9th IEEE working conference on mining software repositories (MSR), pp 50–59

Wan Y, Zhao Z, Yang M, Xu G, Ying H, Wu J, Yu PS (2018) Improving automatic source code summarization via deep reinforcement learning. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering, pp 397–407

Wang S, Lo D (2014) Version history, similar report, and structure: putting them together for improved bug localization. In: Proceedings of the 22nd international conference on program comprehension, pp 53–63

Wang Q, Parnin C, Orso A (2015) Evaluating the usefulness of IR-based fault localization techniques. In: Proceedings of the 2015 international symposium on software testing and analysis, pp 1–11

Wen M, Wu R, Cheung S-C (2016) Locus: locating bugs from software changes. In: 2016 31St IEEE/ACM international conference on automated software engineering (ASE), pp 262–273

Wong C-P, Xiong Y, Zhang H, Hao D, Zhang L, Mei H (2014) Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: 2014 IEEE international conference on software maintenance and evolution, pp 181–190

Wong WE, Gao R, Li Y, Abreu R, Wotawa F (2016) A survey on software fault localization. IEEE Trans Softw Eng 42(8):707–740

Xia X, Lo D, Wang X, Zhang C, Wang X (2014) Cross-language bug localization. In: Proceedings of the 22nd international conference on program comprehension, pp 275–278

Xuan H, Li M (2017) Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: IJCAI, pp 1909–1915

Xuan H, Li M, Zhou Z-H (2016) Learning unified features from natural and programming languages for locating buggy source code. In: IJCAI, pp 1606–1612

Ye X, Bunescu R, Liu C (2014) Learning to rank relevant files for bug reports using domain knowledge. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, pp 689–699

Ye X, Shen H, Ma X, Bunescu R, Liu C (2016) From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th international conference on software engineering, pp 404–415

Youm KC, Ahn J, Kim J, Lee E (2015) Bug localization based on code change histories and bug reports. In: 2015 Asia-pacific software engineering conference (APSEC), pp 190–197

Zhang H (2009) An investigation of the relationships between lines of code and defects. In: 2009 IEEE international conference on software maintenance, pp 274–283

Zhao L, Callan J (2012) Automatic term mismatch diagnosis for selective query expansion. In: Proceedings of the 35th international ACM SIGIR conference on research and development in information retrieva, pp 515–524

Zimmermann T, Premraj R, Bettenburg N, Just S, Schroter A, Weiss C (2010) What makes a good bug report? IEEE Trans Softw Eng 36(5):618–643. https://doi.org/10.1109/TSE.2010.63

Zhou J, Zhang H, Lo D (2012) Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In: 2012 34th international conference on software engineering (ICSE), pp 14–24

Affiliations

Wei Li[1] · **Qingan Li**[1] · **Yunlong Ming**[2] · **Weijiao Dai**[2] · **Shi Ying**[1] · **Mengting Yuan**[1]

Wei Li
weili@whu.edu.cn

Yunlong Ming
mingyunlong@huawei.com

Weijiao Dai
dweijiao@sina.com

Shi Ying
yingshi@whu.edu.com

1 School of Computer Science, Wuhan University, Wuhan, China

2 Huawei Technologies Co., Ltd., Shenzhen, China