7-2023

Duplicate bug report detection: How far are we?

Ting ZHANG
Singapore Management University, tingzhang.2019@phdcs.smu.edu.sg

DongGyun HAN
Singapore Management University, dhan@smu.edu.sg

Venkatesh VINAYAKARAO

Ivana Clairine IRSAN
Singapore Management University, ivanairsan@smu.edu.sg

Bowen XU
Singapore Management University, bowenxu@smu.edu.sg

See next page for additional authors

Citation

Author

Ting ZHANG, DongGyun HAN, Venkatesh VINAYAKARAO, Ivana Clairine IRSAN, Bowen XU, Thung Ferdian, David LO, and Lingxiao JIANG

Duplicate Bug Report Detection: How Far Are We?

TING ZHANG, Singapore Management University
DONGGYUN HAN, Royal Holloway, University of London
VENKATESH VINAYAKARAO, Chennai Mathematical Institute
IVANA CLAIRINE IRSAN, BOWEN XU, FERDIAN THUNG, DAVID LO, and LINGXIAO JIANG, Singapore Management University

Many Duplicate Bug Report Detection (DBRD) techniques have been proposed in the research literature. The industry uses some other techniques. Unfortunately, there is insufficient comparison among them, and it is unclear how far we have been. This work fills this gap by comparing the aforementioned techniques. To compare them, we first need a benchmark that can estimate how a tool would perform if applied in a realistic setting today. Thus, we first investigated potential biases that affect the fair comparison of the accuracy of DBRD techniques. Our experiments suggest that data age and issue tracking system (ITS) choice cause a significant difference. Based on these findings, we prepared a new benchmark. We then used it to evaluate DBRD techniques to estimate better how far we have been. Surprisingly, a simpler technique outperforms recently proposed sophisticated techniques on most projects in our benchmark. In addition, we compared the DBRD techniques proposed in research with those used in Mozilla and VSCode. Surprisingly, we observe that a simple technique already adopted in practice can achieve comparable results as a recently proposed research tool. Our study gives reflections on the current state of DBRD, and we share our insights to benefit future DBRD research.

CCS Concepts: • **Software and its engineering** → Maintaining software;

Additional Key Words and Phrases: Bug reports, duplicate bug report detection, deep learning, empirical study

ACM Transactions on Software Engineering and Methodology, Vol. 32, No. 4, Article 97. Pub. date: May 2023.

97

1 INTRODUCTION

Bug reports (BRs) play an essential role in the software development process, and meanwhile, **issue tracking systems (ITSs)** become necessary to manage BRs. The existence of duplicate BRs may cost extra software maintenance efforts in bug triage and fixing [31]. In practice, duplicate BRs can also be hard to identify. For example, prior research [41] found that there are duplicate BRs taking thousands of days to identify, with up to 230 comments, involving up to 75 people. This kind of BRs consumed much effort before it was finally identified as a duplicate. To alleviate the heavy burden of triagers and decrease the cost of software maintenance, many automatic techniques have been proposed in the past decade [8, 43, 51, 61]. Among them, **Duplicate Bug Report Detection (DBRD)** techniques have been deployed in practice. For example, some ITSs provide recommendations of potential duplicates to a reporter prior to submitting a BR. Others employ a bot that flags a duplicate after it has been submitted. For both scenarios, a ranked list of potential duplicate BRs is produced for manual inspection. Despite the many research works and practitioners' adoption of DBRD, unfortunately, it is unclear which DBRD technique can recommend the duplicate BR most accurately overall. The most recent work by Rodrigues et al. [43] shows that SABD [43] outperforms REP [50] and Siamese Pair [18]. However, their experiments are only limited to a collection of old BRs from Bugzilla ITSs, in which the latest data used belongs to the year 2008. Concurrent with the work by Rodrigues et al. [43], Xiao et al. [57], and He et al. [27] have proposed other DBRD solutions. They have not been compared to each other. Besides, they did not compare with the tools used in practice. This motivates us to: (1) create a benchmark that addresses the limitations of existing evaluation datasets, (2) compare research tools on the same dataset, and (3) compare research and industrial tools.

By studying evaluation data used in the literature, we identify three potential limitations:

— **Age bias:** Firstly, most of the techniques have not been evaluated on the recent BRs. Previous research relies heavily on the dataset proposed by Lazar et al. [32]. This dataset contains BRs from four projects (Eclipse, Mozilla, Netbeans, and OpenOffice) stored in their respective Bugzilla ITSs. All the BRs in these four projects are reported as early as July 1998 and till January 2014. The effectiveness of DBRD techniques on BRs submitted in the year 1998 should be significantly different from their effectiveness on recent BRs. Clearly, a DBRD technique that works well on data from 10 years ago but no longer so on recent data should not be of much use to developers today. We refer to this potential bias as *age bias*.

— **State bias:** Secondly, as indicated by Xia et al. [56], several fields of a BR may change during its lifetime, e.g., summary, version, priority, and so on. Various reasons can lead to changes in BR fields, such as when a bug reporter is new to the open-source project, he might submit a BR where some of the fields are wrongly assigned. Most studies evaluate the effectiveness of DBRD techniques based on the *latest* states of the fields at the point of data collection. The effectiveness of DBRD techniques should be significantly different if the *initial* states of the fields are used. We refer to this potential bias as *state bias*.

— **ITS bias:** Lastly, as these techniques are commonly evaluated on BRs from one specific ITS (i.e., Bugzilla), it is unclear how they perform on other ITSs (e.g., Jira and GitHub). These ITSs are different in the list of fields that they support, and some DBRD techniques make use of fields that exist in Bugzilla but not others. A DBRD technique that is specifically designed for a certain ITS data should perform differently when applied in another ITS. We refer to this potential bias as *ITS bias*.

These biases motivate us to investigate our first research question:

RQ1: *How significant are the potential biases on the evaluation of DBRD techniques?*

To answer RQ1, we investigate the performance of the three best-performing solutions identified by Rodrigues et al. [43] in the presence and absence of each potential bias. We demonstrate that the *age bias* and *ITS bias* matter significantly (p-value < 0.01) and substantially (large effect size) for all but one case, while the impact of *state bias* is insignificant (p-value > 0.05) for all cases.

Based on the above findings, we create a benchmark that addresses age bias and ITS bias and use it to evaluate DBRD techniques that have been shown competitive performance in the research literature. Specifically, we conduct a comparative study that evaluates DBRD techniques on recent BRs (addressing *age bias*) from different ITSs (addressing *ITS bias*). Other than the three techniques mentioned before, we also include two additional recently-proposed DBRD techniques, DC-CNN [27], and HINDBR [57]. We hereby ask our second research question:

RQ2: *How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?*

Our result shows that, surprisingly, for most projects, the retrieval-based approach, i.e., REP [50], proposed a decade ago, can outperform the recently proposed more advanced and sophisticated models based on deep learning by 22.3% on average in terms of Recall Rate at Top-10 positions (a.k.a. RR@10). This again demonstrates the value of simpler approaches in the saga of simple vs. complex [21, 36, 60].

Furthermore, the research tools have been evaluated in isolation ignoring tools that have been used in practice. To address this gap, we investigate our third research question:

RQ3: *How do the DBRD approaches proposed in research literature compare to those used in practice?*

To answer RQ3, we compare the five research tools considered in RQ2 with two tools used by practitioners. The first tool is the DBRD technique implemented in the Bugzilla ITS, named **Full-Text Search** (FTS). It has been deployed in practice by Mozilla [17]. The next tool is the VSCodeBot, which includes a DBRD feature used in Microsoft's VSCode repository [6]. The experimental results with FTS show that a straightforward duplicate search method used in Mozilla can act as a useful baseline as it can outperform the second best-performing technique on one of the projects in our benchmark by 7.6% in terms of RR@10. Still, the best-performing research tools can boost FTS performance by 22.1% to 62.7%. Moreover, the experimental results on VSCodeBot show that VSCodeBot is better than most tools. Still, the two best-performing research tools can outperform VSCodeBot by 7.6% and 9.8% in terms of RR@5. The results show the value of research on DBRD and the need to develop these research tools further so that practitioners can use and benefit from them.

Our main contributions can be summarized as follows:

— *A study about bias in DBRD data.* We investigate the significance of age bias, state bias, and ITS bias in the evaluation of DBRD techniques. The result depicts that age bias and ITS bias have significant and substantial impacts, while state bias has no significant impact.

— *A new benchmark.* We provide a rich dataset containing recent three-year BRs from Bugzilla, Jira, and GitHub ITSs of six projects. We release our replication package including the data and code.[1]

— *A comparative study.* We evaluate state-of-the-art DBRD research tools on a revised dataset that addresses age and ITS biases, and the result shows a simple retrieval-based approach can beat recently-proposed sophisticated deep learning-based models. We also compare state-of-the-art DBRD research tools with two industrial tools, i.e., (1) FTS implemented in Bugzilla ITS and used by Mozilla, (2) VSCodeBot used by VSCode repository. The result shows that

---

[1]https://github.com/soarsmu/TOSEM-DBRD.

FTS outperforms some research tools on some projects. The best-performing research tools however can outperform FTS by 22.1% to 62.7% in terms of RR@10. VSCodeBot is better than most research tools, but the best research tool can outperform it by 9.8% in terms of RR@5.

The rest of the article is organized as follows. Section 2 presents the background information on DBRD. Section 3 describes how we construct the dataset. Section 4 details our experimental setup. Section 5 reports and analyzes the experimental results. Section 6 outlines the novel insights from our experiments. The analysis of threats to validity is covered in Section 7. Section 8 discusses the related work. Finally, we conclude our work and outline the potential future research directions in Section 9.

2 BACKGROUND

Several DBRD techniques have been proposed in research to detect duplicate BRs automatically. Yet, the industry uses a different set of tools. However, there is a lack of a systematic study about different approaches for DBRD. Here, we present an overview of BRs in ITSs, DBRD in practice, and DBRD in research.

2.1 Bug Reports in Issue Tracking Systems

Existing research [43, 57] heavily uses data from Bugzilla for comparing DBRD techniques. We observe differences across ITSs in submitting BRs, finding duplicates, and marking BRs as duplicates. We select the ITSs in the study based on two factors: whether open-source projects can use them and their popularity. Based on the criteria, we choose Bugzilla, Jira, and GitHub.

Bugzilla is one of the world-leading free ITSs. Several large projects such as `Mozilla` and `Eclipse` use Bugzilla. A typical bug reporting process involves providing necessary textual and categorical details. A new BR submission in Bugzilla can be seen as a form-filling activity. To mark a BR as duplicate [19] in Bugzilla, users set the `resolution` field to `duplicate` and insert the bug id that this BR duplicates.

Jira has gained popularity over the last decade. According to the Jira official website [3], more than 65,000 organizations used Jira in 2021. Jira follows a form-filling design similar to Bugzilla for creating a new BR. While creating a BR, Jira allows users to relate existing bugs using labels: `is caused by`, `is duplicated by`, or `relates to`. New duplicates are manually marked using the `resolution` field.

GitHub is a popular Git repository hosting service. It provides rich features, one of which is issue tracking. Issues in GitHub carry a simpler structure when compared to Bugzilla and Jira. Apart from the textual information, the categorical fields are customizable for each repository. Repositories on GitHub can use `label` to categorize issues. However, these labels are not mandatory or well-defined like Bugzilla and Jira. Hence, BRs logged in Bugzilla and Jira have relatively more categorical or structured information. This type of flexibility has pros and cons. On the one hand, it makes it easy to report the issue details; on the other hand, it makes it difficult to extract useful categorical information systematically. Some GitHub repositories provide templates to help users embed categorical information in the textual information. For example, the VSCode project provides a textual template that asks users to provide categorical information such as *VSCode version* and *os version*. However, the textual template is not compulsory to follow and can be ignored by issue reporters. The ITS of GitHub uses labels and comments to mark issues as duplicates. We will describe its duplication marking approach in detail in Section 3.3.

BRs from different ITSs carry some common fields (such as *BugId*, *Created Date*, etc.) and also some different ones (such as *Severity* that exists in BRs from Bugzilla ITS but does not exist in BRs from Jira and GitHub ITSs). We show one example BR from each of the three ITSs in Table 1.

Table 1. Three Example BRs from Eclipse (Bugzilla), Apache (Jira), VSCode (GitHub) Project

Field	Bugzilla	Jira	GitHub
Bug Id	542516	HIVE-21207	92171
Created	2018-12-07 08:01 EST	04/Feb/19 11:02	7 Mar 2020
Product	Platform	-	-
Component	SWT	None	-
Version	4.8	None	-
Priority	P3	Major	-
Severity	major	-	-
Status	CLOSED	RESOLVED	CLOSED
Resolution	DUPLICATE	Duplicate	-
Summary	Unable to upload Image after login	Use 0.12.0 libthrift version in Hive	"C# extension recommended for this file type"
Description	Starting with Eclipse 4.8, horizontal scrolling is ...(omitted)	Use 0.12.0 libthrift version in Hive.	Issue Type: Bug Recently, when I open a .cs file. I get the notification... (omitted)
Ways to Record Duplicate	Field: <dup_id>530693</dup_id>	Issue Links: is duplicated by Bug HIVE-21173; HIVE-21000	Inferred from the comments below: VSCodeBot gave a recommendation, the issue reporter acknowledged
URL:	https://bugs.eclipse.org/bugs/show_bug.cgi?ctype=xml&id=542516	https://issues.apache.org/jira/browse/HIVE-21207	https://github.com/microsoft/vscode/issues/92171

"-" : this field is not available in the ITS; "None" : the value of the corresponding field is empty.

As shown in the table, a Bugzilla BR consists of several fields, the types of which can either be textual or categorical. For categorical fields: Product usually represents a software product that is shipped, and Component is a part of a product. Severity states how severe the problem is, while Priority is used by the bug assignee to prioritize the issues. Status indicates the current state the bug is in, and Resolution indicates what happens to this bug. A BR can have several possible statuses, such as unconfirmed, confirmed, fixed, in process, resolved, and so on. The possible resolution values of a bug can be duplicate, fixed, wontfix, and so on. However, Jira does not provide exactly the same fields. There are no Severity and Product fields in Jira. For GitHub, there are no categorical fields. The textual data mainly includes a summary, description, and so on. All the ITSs contain these two common textual fields. The Summary is usually a one-sentence text describing a bug. The Description usually contains more details, such as the steps to reproduce the bug.

2.2 DBRD in Practice

The existence of duplicate BRs causes increasing software maintenance efforts in bug triage and fixing [31]. Several causes can result in duplicate BRs. Prior work [14] shows that duplicate BRs can either be submitted intentionally (e.g., usually when the reporters are frustrated by the same bug not being resolved), or unintentionally (e.g., reporters do not search for existing BRs or cannot identify duplicates due to the lack of experience). To help lighten the workload of bug triagers and avoid redundant bug fixing, DBRD techniques can be useful in two use scenarios. DBRD approaches can either work for (1) *pre-submission* scenario: They can be integrated into ITSs to make bug reporters aware of existing similar BRs and prevent duplicate BRs from happening. Figure 1 depicts how the JIT duplicate recommendation works at the pre-submission for Mozilla

Fig. 1. An example of duplicate issue recommendation when typing "always loading" before submitting a new BR on Mozilla Firefox.

Fig. 2. An example of VSCodeBot duplicate issue recommendation (issue 75817) from Microsoft/VSCode repository.

ITSs. or (2) *post-submission* scenario: They can recommend duplicate lists after duplicate BRs are submitted. For example, VSCodeBot that is adopted by VSCode GitHub repository can recommend a list of potential duplicates. Like other bots [25], VSCodeBot also communicates with developers via issue comments and pull request comments. Figure 2 shows an example of VSCodeBot duplicate issue recommendation on issue 75817.[2] After a user submits the issue, VSCodeBot detects

---

[2]https://github.com/microsoft/vscode/issues/75817.

Table 2. Comparison between Different Approaches

Approach	Type	Feature Engineering		Distance Measurement
		Embedding	Modeling	
REP [50]	Categorical	-	handcrafted	linear combination
	Textual	-	$BM25F_{ext}$	
Siamese Pair [18]	Categorical	customized	single-layer	Cosine Similarity
	Textual	GloVe	bi-LSTM + CNN	
SABD [43]	Categorical	customized	ReLU	fully-connected layer
	Textual	GloVe	bi-LSTM + attention	fully-connected layer
HINDBR [57]	Categorical	HIN2vec	MLP	Manhattan Distance
	Textual	Word2vec	RNN	
DC-CNN [27]	Categorical	Word2vec	dual-channel CNN	Cosine Similarity
	Textual			

five potential duplicates. The two usage scenarios may require different technical considerations. For the pre-submission usage scenario, the efficiency of DBRD tools plays an important role as it requires DBRD tools to produce a real-time recommendation. Issue reporters are unlikely to be willing to wait for a long time for a DBRD tool to return some results. On the other hand, for the post-submission usage scenario, the DBRD tools can potentially be run overnight in a batch mode to process many BRs, and the emphasis is on optimizing accuracy.

2.3 DBRD in Research

Here, we present an overview of the DBRD research tools considered in our work. In the research literature, many DBRD approaches have been proposed and evaluated. In our work, we consider five approaches. They include the three best performers in the study conducted by Rodrigues et al. [43]: SABD [43], REP [50], and Siamese Pair [18]. REP is a popular information retrieval-based DBRD approach. Siamese Pair is the first deep learning-based DBRD approach proposed in the literature. We further include HINDBR [57] and DC-CNN [27], which are proposed recently. Since DBRD models mainly differ in how they conduct feature engineering and how they measure similarity between BR pairs, we distinguish these two aspects between different methods in Table 2. For detailed explanations of each model, we refer readers to check the original articles. We describe the five approaches in the following paragraphs, respectively.

REP [50] is a retrieval function to rank BRs based on their similarity with an incoming BR. REP considers seven features, including two textual features, i.e., summary and description fields, and five categorical features, i.e., product, component, type, priority, and versions. The similarity between the two BRs is a weighted linear combination of the scores of the seven features. Specifically, REP represents texts with uni-grams and bi-grams and calculates the textual similarity by computing an extended version of BM25F [42]. Among the five categorical features, if the values of the product, component and type fields of the two BRs are the same, the corresponding feature is 1; otherwise, the feature value is 0. For the remaining two features, i.e., priority and version, the corresponding feature is represented by the inverse of the distance between the values of priority and version of the two BRs. REP has a total of 19 parameters to be tuned, such as the weights of features used. REP proposes to learn these parameters for the bug repositories under consideration using stochastic gradient descent by analyzing a training dataset of historical BRs. The training set of REP is a set of triples (q, rel, irr), where q is the query bug, rel is a duplicate bug with q, and irr is a non-duplicate bug with q.

Siamese Pair [18] is a deep learning architecture combining Siamese LSTM and CNN for DBRD. Siamese Pair first encodes different types of features separately. Specifically, for textual fields, the summary is encoded by a bidirectional LSTM, and the description is encoded by a CNN. The

categorical fields, such as product, priority, and component, are encoded by a feed-forward neural network. In the end, the outputs from the three encoders would be concatenated to represent a BR. The proposed model uses Siamese neural networks and is trained with a max-margin objective. In the evaluation stage, BRs in the test dataset are sorted based on the Cosine similarity with the encoding of the query BR.

Soft Alignment Model for Bug Deduplication (SABD) [43] receives a pair of BRs, a query BR, and a candidate BR. SABD is composed of two sub-networks: one for categorical information and the other for textual information. Each sub-network represents one type of information separately and uses a comparison layer to produce a comparative representation of the two BR vectors either in terms of the values of their categorical or textual fields, respectively. The categorical sub-network is a straightforward dense neural network, while the textual sub-network adopts a sophisticated architecture, where the core is the soft alignment comparison layer. The outputs of the two sub-networks are concatenated. A classifier layer receives the concatenated output and produces the final predicted probability regarding whether the candidate BR is the duplicate of the query BR. The soft-attention alignment is used to exchange the information between BRs before encoding the textual fields into a fixed-size vector.

HINDBR [57] represents BRs as a **Heterogeneous Information Network (HIN)**. The BRs are connected through their categorical fields. For example, two BRs with bug id x and y that have the same priority, say *Pri_High*, are connected as $x - Pri\_High - y$ in the HIN. Manhattan distance on the HIN2Vec [20] embedding is used to find semantic similarity in BRs. Xiao et al. introduced three variations of HINDBR: (1) only use unstructured features (i.e., text), (2) only use structured features (i.e., categorical information), and (3) use both. In this study, we found that the models only use unstructured features show better performance than both, thus, we adopted the variant with only text.

Dual-Channel Convolutional Neural Networks (DC-CNN) [27] represent a BR pair with dual-channel matrices. DC-CNN extracts the values of four fields in a BR, i.e., product, component, summary, and description, and treats them as text. After the pre-processing, including tokenization, stemming, and stop word removal, a word2vec [33, 45] model is learned to capture semantic information in the BRs. DC-CNN converts each BR from text to a single-channel matrix based on the word representations learned by the word2vec model. To represent a BR pair, it combines two single-channel BR representation matrices into dual-channel matrices. The BR pair representations are then fed into a CNN model to capture the correlated semantic relationships between BR pairs. The output of the last layer of the model is used as the predicted similarity score between two BRs.

REP, SABD, and Siamese Pair were evaluated using a *ranking* setting: given a new BR, rank the potential duplicate BRs. HINDBR and DC-CNN were evaluated using a *classification* setting: given a pair of BRs, decide whether they are duplicates. Rodrigues et al. [43] claim that this classification setting is quite unrealistic since the real scenario presents a much larger set of negative candidates. Furthermore, when a new BR is submitted, all previously submitted reports are duplicate candidates. Thus, they believe the classification-setting highly overestimated performance. Following them and considering how DBRD is used in practice (c.f. Section 2.2), we evaluate all these approaches using the *ranking* setting.

3 DATA COLLECTION METHODOLOGY

Different ITSs represent BRs in different ways, and it is essential to prepare a unified data format to represent the BRs extracted from all of these ITSs. We present our methodology to build the BR dataset from three different ITSs. The methodology involves five main steps: (1) crawling BRs from the website, (2) filtering out open/unresolved BRs, (3) extracting duplicate BR relations, (4)

extracting both textual and categorical information, and (5) cleaning and generating duplicate pairs. We elaborate on these steps in the following subsections.

3.1 Data Source Selection and Crawling

Project Selection. From each of the three ITSs, we have selected two projects. Here, we describe the projects selected and the rationale behind the selection.

Bugzilla: We choose `Eclipse` [16] and `Mozilla` [17] for Bugzilla, which have the two largest number of issues in the dataset of Lazar et al. [32]. `Eclipse` is an **Integrated Development Environment (IDE)**, and its Bugzilla contains several products, including C Development Tools, Eclipse Modeling Framework, and so on. Similarly, the Mozilla software foundation also includes several projects, including the popular Firefox web browser.

Jira: Following Xie et al. [58], we select two Apache projects hosted on Jira,[3] i.e., `Hadoop` [2] and `Spark` [5]. Hadoop provides a big data framework for distributed data storage and processing. There are several projects that have been categorized as Hadoop in Jira.[4] In this work, we include `Hadoop Common`, `Hadoop HDFS`, `Hadoop MapReduce`, `Hadoop YARN`, `HBase`, `Hive`, and `Hadoop Development Tools` projects. We collectively call them as `Hadoop`. `Spark` is an analytics engine for large-scale data processing.

GitHub: From the datasets listed at GHtorrent [26], we choose the repositories having the largest number of issues. We use the latest update of this `GHtorrent` dataset dated March 6, 2021. Then we manually confirm whether each repository is still in active use. After excluding test or unavailable repositories, we got the five repositories that contained the largest number of issues. They are `nixos/nixpkgs`, `microsoft/vscode`, `elastic/kibana`, `kubernetes/kubernetes`, and `ansible/ansible`. Among these five repositories, `VSCode` [6] and `Kibana` [4] contained the largest number of duplicate issues in 2018–2020, so we included these two projects in our dataset. **Visual Studio Code (`VSCode`)** is a popular multi-platform source-code editor provided by Microsoft. `Kibana` is a proprietary data visualization dashboard software for Elasticsearch, which is a search engine based on Lucene.

Time Range Selection. We select two different time periods (i.e., old and recent) to investigate whether the age of data impacts the performance of DBRD techniques. For *old* data, we choose January 1, 2012 to December 31, 2014. For *recent* data, we include the recent three-year data from January 1, 2018 to December 31, 2020. The DBRD feature described in Section 2.2 was introduced to Bugzilla in 2011. Moreover, the feature adoption date for the projects is unclear. As we would like to focus on analyzing the impact of *age bias* (rather than the impact of DBRD feature introduction), we intentionally picked the time range after this feature was introduced to Bugzilla. Intuitively, if a significant age bias exists considering a 6-year gap period, the bias would have been more pronounced if a longer gap period is considered.

Crawling. For both Bugzilla and Jira, we used the XML export API of the ITSs. For GitHub issues, we used GraphQL API [1] for retrieving the issues in JSON format. Both XML export API and GraphQL API are publicly available. We crawl the issues in June 2021, which is six months later than the aforementioned recent data, to minimize the number of open or unresolved issues.

---

[3]Although Jira is a proprietary bug tracking and project management software, it is free for open-source projects (e.g., the Apache Software Foundation projects) that meet certain criteria [11].
[4]https://issues.apache.org/jira/secure/BrowseProjects.jspa?selectedCategory=10292.

3.2 Filtering Out Open/Unresolved BRs

Following Lazar et al. [32], we only keep *closed* or *resolved* BRs among all the crawled BRs. A typical life cycle of a bug can be abstracted into six main steps [54]: unconfirmed, new, in progress, resolved, verified, and closed. Note that a *resolved* or *closed* bug can be reopened in the future. Even so, the detailed life cycle in different ITSs may be different. Based on the definition of open bugs by Mozilla,[5] we consider a BR as closed if (1) its status is either resolved or verified; or (2) its resolution is one of the seven types: fixed, invalid, wontfix, moved, duplicate, worksforme, and incomplete. For Eclipse, we add another possible status CLOSED as closed bugs based on the Eclipse Wiki.[6] For Jira projects, a closed bug has much more possible resolution types than Bugzilla. Thus, we regard a BR as closed if its *status* is marked as either resolved or closed. For GitHub projects, as it only has *state* field, other than resolution and status. We consider an issue as closed, if its state is closed.

3.3 Identifying Duplicate BRs

The process to identify whether a BR is a duplicate one is referred to as *duplicate detection*. We use the following strategies to extract ground truths:

Bugzilla: The XML format of each issue in Bugzilla has a field called dup_id, which contains a reference to a bug that the current bug is a duplicate of. The dup_id can either be empty or contains another bug's bug_id. If it is not empty, the current bug is a duplicate of the bug in dup_id. Therefore, we directly extract the *dup_id* of those bugs that have a resolution of DUPLICATE to gather the duplicate relation.

Jira: Jira does not provide the dup_id like Bugzilla does. However, Jira provides rich types of issue dependency links, e.g., duplicates, is duplicated by, contains, to represent the relationship between issues. We identify the duplicate relations by analyzing the duplicates, and is duplicated by links.

GitHub: Other than the prior two ITSs, the issues in GitHub have a more flexible format. GitHub supports marking duplicates with a comment [24] of the format "Duplicate of #ISSUE_NUMBER". Such comment format can be used to mark the issue that is a duplicate of another issue. As it is not mandatory, we found that not all issue reporters strictly follow this comment format pattern, some users use short-hand notations to mark duplicates, for example, *dup with #ISSUE_NUMBER*. In the end, we use a regular expression that can check the variations of duplicate comments to detect duplicate issues. To validate the reliability of our regular expressions, we manually investigated the duplicate issues detected by the regular expression. We randomly sampled 384 duplicate issue pairs based on the regular expressions from Kibana and VSCode, respectively. Then, two authors (i.e., investigators) evaluated independently whether the extracted duplicate issues are real duplicates. If there is a disagreement between the investigators, they discuss their investigation results until they reach an agreement. We found only 3 and 21 cases were wrong (i.e., the extracted issues based on the regular expression are not real duplicates) for Kibana and VSCode, respectively. Thus, our regular expression can detect duplicate issues with at least 94.5% of accuracy. Note we do not check false negatives. As for projects which use Bugzilla and Jira as ITS, false negatives are also possible since people may not mark the duplicates as such.

---

[5]https://wiki.mozilla.org/BMO/UserGuide/BugStatuses.
[6]https://wiki.eclipse.org/Bug_Reporting_FAQ#What_is_the_life_cycle_of_a_bug_report.3F.

Table 3. Textual and Categorical Fields that are Leveraged by the Approaches

Fields		REP [50]	Siamese-Pair [18]	SABD [43]	HINDBR [57]	DC-CNN [27]
Textual	summary	✓	✓	✓	✓	✓
	description	✓	✓	✓	✓	✓
Categorical	product	✓	✓	✓	✓	✓
	component	✓	✓	✓	✓	✓
	priority	✓	✓	✓	✓	
	severity		✓	✓	✓	
	type	✓				
	version	✓			✓	

3.4 Information Extraction

All the selected DBRD approaches have taken advantage of both textual and categorical information to improve their effectiveness. Table 3 shows the fields that each approach can utilize. The three ITSs have different fields to record categorical information. We extract all the essential fields needed by the approaches, i.e., bug id, categorical fields: `product`, `component`, `severity`, `priority`, `version`, `status`, `resolution`; textual information: `summary`, `description`. We describe how we extract such information from different ITSs.

Bugzilla: A BR on Bugzilla that is exported as an XML file contains clear and well-defined field names. Around 20 fields are given in the XML file, however, not all of them are needed for DBRD. We thus parsed the XML file and saved the essential fields.

Jira: Similar to Bugzilla, Jira has several pre-defined fields for bug reporters to fill in, e.g., `component`, `priority`, `version`. However, Bugzilla and Jira do not have identical categories. For instance, Jira only has a `priority` field to indicate the importance of an issue, while Bugzilla has `severity` and `priority`. Different from Bugzilla, BRs on Jira do not have the `product` and `severity` fields. If any field value is missing, we leave it as an empty string.

GitHub: As mentioned before, GitHub does not provide any well-defined fields for categorical information. Even though labels may provide categorical information, labels are highly customizable for each project. In addition, labels are shared by the other features (e.g., pull requests and discussion) of GitHub,[7] so it does not represent categories for issues only. Due to the above limitations, we only extracted textual information from GitHub issues. We leave it as future work to investigate how to derive categorical information from GitHub issues.

3.5 Data Cleaning

Handling duplicates. It is not rare that one BR has more than one duplicate BRs. For example, the bug 92250 has 45 duplicates.[8] A *group* or *bucket* refers to a set of BRs which are duplicates to each other. The *master* BR is the one to which the rest reports refer. Similar to the prior work [43, 50], we also make the first submitted BR as the *master* and the rest in the bucket as *duplicates*. As we do not cover all the BRs in each ITS and only use the time range of three years, the master BRs of some duplicate BRs can be out of our considered time range (i.e., before January 1, 2012, or January 1, 2018). In that case, we choose the oldest BR in the bucket within the time range as a new master. By this cleaning step, the number of duplicates in our dataset has been smaller than the number of BRs that have been resolved as duplicates.

---

[7]https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/managing-labels.
[8]https://bugs.eclipse.org/bugs/duplicates.cgi.

Table 4. Statistics of Data in the Six Projects

ITS	Project	# BRs	# Dup BRs (%)	# Unique Master BRs	Per bucket	
					Avg. BRs	Max. BRs
Bugzilla	Eclipse	27,583	1,447 (5.2%)	959	2.5	19
	Mozilla	193,587	20,189 (10.4%)	10,702	2.9	151
Jira	Hadoop	14,016	377 (2.7%)	336	2.1	6
	Spark	9,579	354 (3.7%)	290	2.2	14
GitHub	Kibana	17,016	470 (2.8%)	388	2.2	9
	VSCode	62,092	4386 (7%)	2,342	2.9	51

Avg. BRs: average number of BRs in a bucket; Max. BRs: max number of BRs in a bucket.

Basic pre-processing. We conducted data pre-processing in the textual part (summary and description) of a BR: (1) removing punctuations, (2) lower-casing, (3) removing numbers, (4) removing stop words, (5) stemming, and (6) removing single characters. We then tokenized the processed text with white spaces. We reused the script in the replication package provided by Rodrigues et al. [43].

Train-test split. We first sort all the BRs chronologically. Then, we select three years of data. The first two years of data are used for training (including validation, if applicable), while the last year of data is used for testing. Except for REP, all the other approaches need validation data. SABD and Siamese Pair use the last 5% data in training data as validation data, while HINDBR and DC-CNN use 20% and 10% of the training pairs as validation pairs, respectively.

One-year time window. For each duplicate BR in the test data, its candidate duplicates are the BRs submitted within one year. We call it *one-year time window* search range. Rodrigues et al. found that despite some minor differences, the findings using a one-year time window are similar to the ones with a longer frame of three years. Thus, we decide to use a one-year time window in our experiments.

Pairs generation. Generating duplicate and non-duplicate pairs is an important implementation detail integrated into each approach. By design, different approaches utilize different ratios of non-duplicate and duplicate pairs. And we follow the original implementation of each approach. Specifically, REP generates 30 times more non-duplicate pairs than duplicate pairs. Siamese Pair and SABD generate the same number of non-duplicate pairs as duplicate pairs. HINDBR and DC-CNN generate four times more non-duplicate pairs than duplicate pairs. However, the duplicate pairs in the training data are the same, i.e., pair two BRs that are in the same bucket. We report the number of duplicate pairs in training data in Table 4. Note that we would only pair those BRs in the training set.

Overall, we present the basic statistics of the six projects in Table 4. We can find that different projects have different characteristics: in one bucket, Mozilla can have 151 BRs that are duplicates of each other, while for Hadoop, the maximum BRs in one bucket is only 6. The ranking of the total number of BRs in each ITS is Jira < GitHub < Bugzilla. On average, the percentage of duplicate BRs is also Jira < GitHub < Bugzilla.

4 EMPIRICAL SETTINGS

4.1 Experimental Setup

We design experiments to answer the RQs described in Section 1. For RQ1, we focus on understanding the biases that may affect the performance of DBRD techniques. For RQ2, we conduct a comparison among the existing DBRD techniques on our new benchmark. RQ3 focuses on the DBRD techniques in practice.

Table 5. Statistics of Old (2012–2014) and Recent (2018–2020) Data for RQ1

Project	Age	Train		Test	Total	
		# BRs (% Dup)	# Dup Pairs	# BRs (% Dup)	# BRs (% Dup)	# Master BRs
Mozilla	Old	198,653 (9.9%)	35,474	139,502 (9.9%)	338,155 (9.9%)	21,554
	Recent	137,886 (10.1%)	60,498	55,701 (11.2%)	193,587 (10.4%)	10,702
Eclipse	Old	49,355 (5.5%)	4,482	25,021 (12.1%)	74,376 (7.7%)	3,254
	Recent	19,607 (4.7%)	1,725	7,976 (6.5%)	27,583 (5.2%)	959

Table 6. The Percentage of BRs Changed the Corresponding State in 2018–2020

Platform	Summary	Description	Product	Component	Priority	Severity	Version
Eclipse	10.8%	-	7.8%	11.7%	1.2%	5.6%	8.6%
Mozilla	11.8%	-	21.4%	24.5%	24.5%	5.4%	4.2%

RQ1: To answer RQ1, we evaluate the three best-performing DBRD techniques reported in a recent study by Rodrigues et al. [43]: REP [50], Siamese Pair [18], and SABD [43]. For the age bias (old vs. recent data) and state bias (initial vs. latest state), we run experiments on the BRs from Bugzilla. For the ITS bias, we run experiments on the BRs from Bugzilla, Jira, and GitHub (Bugzilla vs. Jira, and Bugzilla vs. GitHub). To analyze the impact of age bias and state bias, we pick two popular projects (Mozilla and Eclipse) which use Bugzilla as their ITS as a starting point. We conduct controlled experiments [13, 22] by varying one variable (i.e., age and state) at a time. Furthermore, We choose Mozilla and Eclipse projects due to the long history associated with them in terms of both the number of BRs and the number of duplicates.

To investigate the impact of *age bias*, we evaluate the effectiveness of the three DBRD techniques on BRs from two time windows: old (2012–2014) vs. recent (2018–2020). Table 5 shows the data statistics of these two time windows.

To investigate the impact of *state bias*, we use Mozilla and Eclipse BRs that were submitted in 2018–2020. For these BRs, we compare the effectiveness of the three DBRD techniques using the latest and initial states of their fields. Among the fields we extracted for experiments, only the description cannot be changed [52], thus, we try to recover all the other fields if changed. We recover the initial state of these fields by tracing BR's change history.[9] Each item in the change history of a BR describes the author, time, updated field, removed value, and added value. Table 6 shows the percentage of BRs which changed their initial states.

To investigate the impact of *ITS bias*, we evaluate the three DBRD techniques on BRs (reported in 2018–2020) for three sets of projects that use Bugzilla (Eclipse and Mozilla), Jira (Spark and Hadoop), and GitHub (Kibana and VSCode) as ITS.

RQ2: To answer RQ2, other than the three techniques evaluated for answering RQ1, we add two more recent techniques, DC-CNN [27] and HINDBR [57]. These two approaches are initially designed and also evaluated as a classification task. In our experimental setting, all the BRs submitted within a one-year time window are duplicate BRs candidates. For each BR $br_i, i = 1, \ldots, m$, where m is the total number of BRs in the test set. We pair it with all its candidate BRs, i.e., $(br_i, br_{i,1}), (br_i, br_{i,2}) \ldots (br_i, br_{i,k})$, where k is the total number of candidate BRs of br_i, $br_{i,j}, j = 1, \ldots, k$ is a candidate duplicate of br_i. The trained classification model is used to predict whether each pair is duplicate. The output probability value is then used to rank the possibility that each candidate BR be a duplicate one. If two candidate BRs have the same probability value, we rank them based on their BR ids in ascending order. We get the top-k recommendations

---

[9]An example of change history of a BR: https://bugzilla.mozilla.org/show_activity.cgi?id=122876.

Table 7. The Number of Issues Per Duplicate Recommendation
Frequency by VSCodeBot

# predictions	1	2	3	4	5	Total
# issues	5,016	2,006	1,158	730	2,898	11,808

and evaluate the performance based on the recommendations. We evaluate the five techniques on a new benchmark dataset unaffected by the biases that are found to have a significant and substantial impact on DBRD evaluation in RQ1.

RQ3: To answer RQ3, we investigate the effectiveness of the DBRD techniques used in practice by Mozilla and VSCode projects, and compare them with the aforementioned five DBRD techniques.

Mozilla uses Bugzilla as its ITS, and Bugzilla implements several variants of a DBRD technique named FTS.[10] We study the source code of Bugzilla to identify how FTS works. Simply put, based on the summary input, FTS relies on a BR database and issues SQL queries to search in the database. There are two variants of FTS: FULLTEXT_OR and FULLTEXT_AND. For the first variant, the SQL queries specify "OR" operations in FTS; otherwise, the SQL queries specify an exact match. We inferred the variant of the FTS search used by Mozilla (i.e., the OR variant) by trying to enter new BRs into its FTS. We replicated the OR-variant by using the same SQL queries that Bugzilla uses on reading the entire summary field.

Another DBRD technique used in practice is VSCodeBot [7], but its implementation is not publicly available, and it only works for the VSCode repository. Thus, we only evaluate the five techniques on the test data from the VSCode project. Table 7 presents the number of VSCode issues that have potential duplicate recommendations by VSCodeBot made in 2018–2020. Among the 62,092 VSCode BRs in our dataset, 11,808 BRs got potential duplicate recommendations. VSCodeBot only recommends up to five recommendations for an issue.

4.2 Evaluation Metrics

We evaluate the effectiveness of DBRD tools by calculating Recall Rate@k (RR@k). The evaluation strategy is consistent with the previous works for the DBRD task [9, 18, 46, 51, 53], i.e., only RR@k is used. Note that another metric that has been used in a few DBRD research articles [43, 50] is **Mean Average Precision (MAP)**. MAP concerns the position of the ground truth master BR in each prediction. A higher MAP means that for each BR in the test set, the model can return the ground truth master at a higher place in the ranked result list. The primary reason that prior works, as well as our work, do not consider MAP is that it does not simulate the real usage scenario: In real use, it is unlikely developers would frequently check the recommendations after 10 BRs. For example, prior work on understanding practitioners' expectations of automated fault localization [30] has shown that nearly all the respondents (close to 98%) are unwilling to inspect more than ten program elements to find the faulty code. Another supporting evidence is for VSCodeBot [7] used in the Microsoft Visual Studio Code repository, the largest number of duplicate issue recommendations is 5.

RR@k is defined as follows:

$$RR@k = \frac{n_k}{m},$$

where n_k is the number of duplicate BRs in the test set whose bucket has been found in the top-k positions; m is the total number of duplicate BRs in the test set.

---

[10]https://github.com/bugzilla/bugzilla/blob/5.2/Bugzilla/Bug.pm   and   https://github.com/bugzilla/bugzilla/blob/5.2/Bugzilla/DB.pm.

test BR1	1	2	3	4	5	6	7	8	9	10
test BR2	1	2	3	4	5	6	7	8	9	10
test BR3	1	2	3	4	5	6	7	8	9	10
test BR4	1	2	3	4	5	6	7	8	9	10

Fig. 3. Examples of the predictions in the top-10 positions for four test BRs.
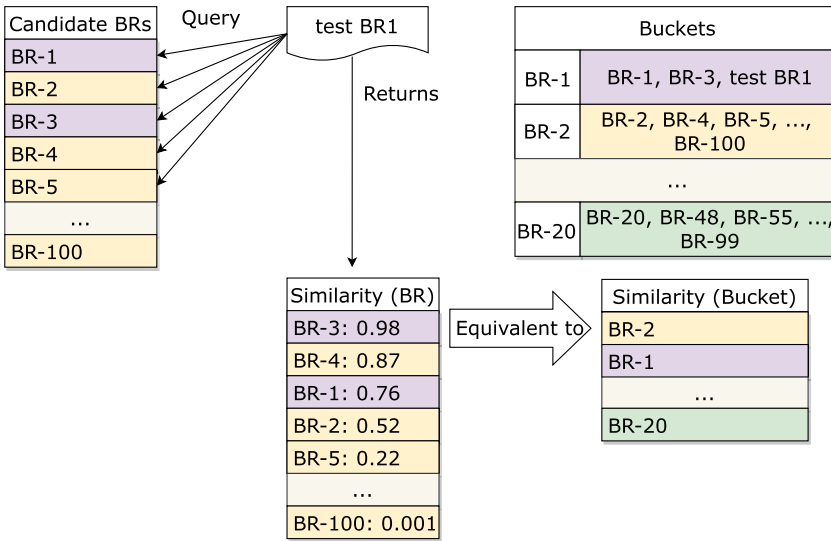
Fig. 4. The workflow of retrieving the correct bucket.

For example, in Figure 3, consider a project with four BRs in the test set. We show the top-10 predictions for the four test BRs. If the corresponding prediction is correct, the box is highlighted in red. In this example, for the test BR1, BR2, and BR3, the successful prediction is in the 2nd, 4th, and 8th predictions, respectively. For the test BR4, all the top-10 predictions are wrong. The RR@k in this dataset would be: RR@$k(k = 1) = 0$, RR@$k(k = 2, 3) = 1/4 = 0.25$, RR@$k(k = 4, 5, 6, 7) = 2/4 = 0.5$, and RR@$k(k = 8, 9, 10) = 3/4 = 0.75$.

The process of finding a duplicate BR's master BR is essentially equivalent to finding the bucket to which it belongs. In the prediction stage, for each BR in the test dataset, a DBRD technique queries its candidate BRs and returns a possible duplicate BR list. In the evaluation stage, we group the BRs in the list returned by a DBRD technique into *buckets* list to calculate the recall rate of the predictions. This evaluation strategy is consistent with Rodrigues et al. [43].

In this paragraph, we elaborate on how n_k in RR@k is calculated in the example in Figure 3. There are four BRs in the test set. To find the BRs that are duplicates of a test BR (i.e., the bucket they belong to), instead of only calculating the similarity score of the test BR with master BRs, we compare the test BR with all the BRs in buckets. Like the prior works [43, 50], we also use the highest score among the test BR with all the BRs in the candidate bucket as the similarity score between the BR with the candidate bucket. We illustrate a concrete example as shown in Figure 4. We have 100 BRs: $\{BR\text{-}1, BR\text{-}2, BR\text{-}3, \ldots, BR\text{-}100\}$ and they belong to 20 buckets

BR-1 : {BR-1, BR-3, $testBR$1}, BR-2 : {BR-2, BR-4, BR-5, . . . , BR-100}, BR-3: {BR-3, . . .}, . . . BR-20: {BR-20, BR-48, BR-55, . . . , BR-99}. We represent a bucket as a dictionary: the key is the master BR, and the value is its duplicate BRs and itself. Thus, we use the master BR id to refer to the bucket. Given $testBR$1, each model will return a list of BRs sorted by the likelihood of being a duplicate of test BR1. The returned rank list is: [(BR-3, 0.98), (BR-4, 0.87), (BR-1, 0.76), (BR-2, 0.52), (BR-5, 0.22), . . . (BR-100, 0.001)]. The first part of each tuple is the BR id, and the second part is the similarity score. According to the ground truth information, test BR1 belongs to the bucket BR-1: {BR-1, BR-3, $testBR$1}. As test BR1 is in the same bucket as BR-1 and BR-3, the highest similarity scores, i.e., (BR-3, 0.98), will be the similarity score between test BR1 with the bucket BR-1: {BR-1, BR-3, $testBR$1}. So in this example, this model makes a successful prediction at the second position and we can get $n_k = 1(k = 2)$. With the same strategy, we can calculate the n_k in the rest three BRs and get RR@$k(k = 2) = 1/4 = 0.25$.

5 EMPIRICAL RESULTS AND ANALYSES

5.1 RQ1. How Significant are the Potential Biases on DBRD Techniques?

Due to the page constraint, we only present the statistical test results in Table 8. The detailed results are available in our online appendix.[11]

Age Bias: We run the Mann–Whitney U test [23] on the following null hypothesis for each pair of DBRD approach and project:

$H_{0.1}$: There is no significant difference in RR@k on old issues and recent issues.

We also compute Cliff's delta (d) effect size [44]. As we have six p-values on the same hypothesis, we also run Bonferroni correction [55], and the significance level α becomes 0.0083. We find that the p-value is <0.0083 for all approach-project pairs except one case (and thus we can reject the null hypothesis) and the effect size is large. We contend that the age of data significantly affects DBRD performance.

State Bias: Similar to the prior bias, we run the Mann–Whitney U test on the following null hypothesis for each pair of the DBRD approach and project:

$H_{0.2}$: There is no significant difference in RR@k on the initial state and recent state within an issue.

We also compute Cliff's delta effect size. With Bonferroni correction, the significance level α is 0.0083. We find that the p-value is > 0.0083 for all approaches on projects with the initial and latest state (and thus we cannot reject the null hypothesis). We contend that the state does not make a significant difference in DBRD performance.

ITS Bias: we also run the Mann–Whitney U test on the following null hypothesis for each approach on Bugzilla vs. Jira, and Bugzilla vs. GitHub data:

$H_{0.3}$: There is no significant difference in RR@k on Bugzilla issues and other ITS issues.

We also compute Cliff's delta effect size. We find that the p-value is < 0.0083 (with Bonferroni correction) for all approaches and the effect size is large (except for one case), and thus we can reject the null hypothesis. Thus, we contend that ITS plays an important role in DBRD technique performance.

Answer 1: Age bias *has a statistically significant impact (with large effect size) on the evaluation of DBRD techniques in all but one cases.* State bias *does not have a statistically significant impact.* ITS bias *has a statistically significant impact (with large effect size) on all but one case.*

---

[11]https://github.com/soarsmu/TOSEM-DBRD.

Table 8. Mann–Whitney-U with Cliff's Delta Effect Size $|d|$ on RQ1

Bias	Approach	Data	p-value	$|d|$
Age	REP	Eclipse	0.003	0.78 (large)
		Mozilla	0.005	0.72 (large)
	Siamese Pair	Eclipse	<0.001	1 (large)
		Mozilla	0.003	0.76 (large)
	SABD	Eclipse	0.001	0.82 (large)
		Mozilla	0.012	0.66 (large)
State	REP	Eclipse	0.105	0.44 (medium)
		Mozilla	0.190	0.36 (medium)
	Siamese Pair	Eclipse	0.063	0.5 (large)
		Mozilla	0.190	0.36 (medium)
	SABD	Eclipse	0.315	0.28 (small)
		Mozilla	0.315	0.28 (small)
ITS	REP	Jira	0.056	0.36 (medium)
		GitHub	<0.001	0.66 (large)
	Siamese Pair	Jira	<0.001	1 (large)
		GitHub	<0.001	0.97 (large)
	SABD	Jira	<0.001	0.97 (large)
		GitHub	<0.001	0.77 (large)

Table 9. Statistics of Training and Testing Data

ITS	Project	Train		Test	Total
		# BRs (% Dup)	# Dup Pairs	# BRs (% Dup)	# BRs (% Dup)
Bugzilla	Eclipse	19,607 (4.7%)	1,725	7,976 (6.5%)	27,583 (5.2%)
	Mozilla	137,886 (10.1%)	35,474	55,701 (11.2%)	193,587 (10.4%)
Jira	Hadoop	10,276 (2.8%)	328	3,740 (2.5%)	14,016 (2.7%)
	Spark	6,738 (4%)	414	2,841 (3%)	9,579 (3.7%)
GitHub	Kibana	9,849 (2.9%)	376	7,167 (2.6%)	17,016 (2.8%)
	VSCode	40,801 (7.2%)	9,008	21,291 (6.8%)	62,092 (7%)

5.2 RQ2. How do the State-of-the-art DBRD Research Tools Perform on Recent Data?

Based on the findings from RQ1, we evaluate the existing DBRD techniques on a new benchmark that omits *age bias* and *ITS bias*. Our benchmark contains the recent three-year BRs extracted from Bugzilla, Jira, and GitHub. We build this dataset as described in Section 3. Table 9 shows the statistics of the training and testing data. Figure 5 shows RR@k from the five approaches on our dataset. The x-axis denotes k values from 1 to 10 while the y-axis shows RR@k. Each approach is highlighted with a different shape and color. For all datasets, REP outperforms the other approaches except for Mozilla and VSCode. On average, REP outperforms SABD by 22.3% in terms of RR@10 across six project data. As presented in Table 9, both Mozilla and VSCode have the largest number of BRs and duplicates BRs. For these two projects, SABD shows comparable results with REP and even outperforms on VSCode dataset by 9% in terms of RR@10. Siamese Pair, HINDBR, and DC-CNN show fluctuating results. Siamese Pair presents higher RR@k values on Eclipse, Mozilla, and VSCode, while it demonstrates lower RR@k than HINDBR and Hadoop on Spark and Kibana.

Component Analysis. As shown in Figure 5, REP is the winner in five out of the six datasets and it takes advantage of the information from most of the fields. Thus, we investigate REP to understand the contribution of each component in DBRD. REP initializes the weights of the textual

(a) Eclipse

(b) Mozilla

(c) Hadoop

(d) Spark
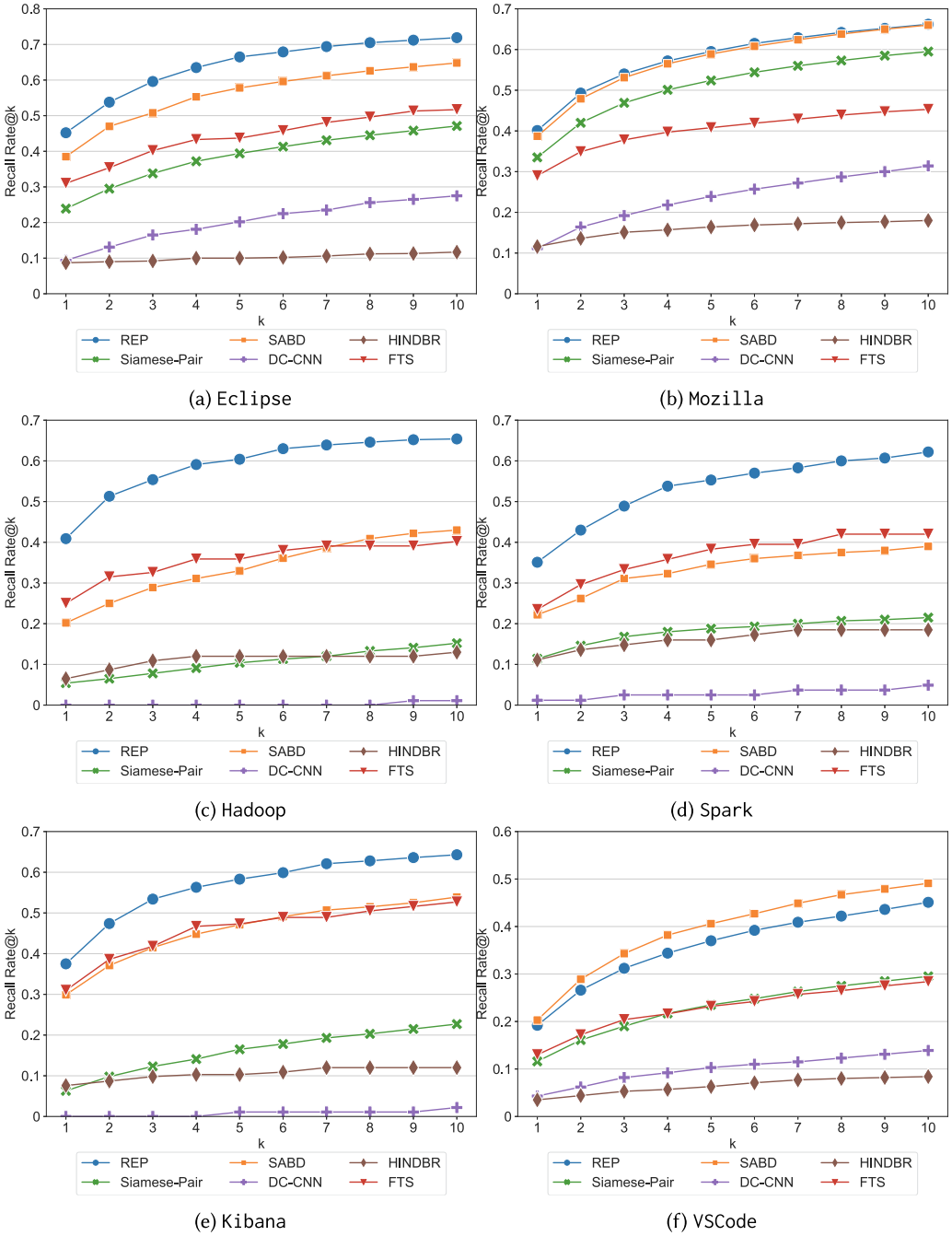
(e) Kibana

(f) VSCode

Fig. 5. Recall Rate@k in the test data of Eclipse, Mozilla, Hadoop, Spark, Kibana, and VSCode.

features higher than the rest of the features. Additionally, the initial weight of the summary is also higher than the weight of the description. Besides, among the total 19 parameters tuned by gradient descent, 14 parameters relate to the textual fields. Based on the observation, our assumption is that textual fields are the most crucial components among all the fields considered. To further verify our

Table 10. Investigation of which Component Benefits REP

RR@k	All	w/o				
		description	short_desc	product	component	priority
1	0.460	0.327	0.350	0.456	0.458	0.450
2	0.544	0.415	0.458	0.527	0.554	0.540
3	0.610	0.456	0.494	0.575	0.598	0.602
4	0.646	0.490	0.510	0.617	0.633	0.637
5	0.673	0.515	0.533	0.644	0.658	0.665
6	0.690	0.531	0.552	0.662	0.663	0.679
7	0.704	0.544	0.569	0.671	0.671	0.694
8	0.706	0.556	0.579	0.681	0.683	0.706
9	0.715	0.565	0.600	0.687	0.683	0.712
10	0.721	0.573	0.613	0.698	0.692	0.717

hypothesis, we investigate the contributions of each field value in REP. We run REP on the Eclipse dataset and each time set a certain field value as empty. Table 10 demonstrates the performances when we set each field as empty. We can find that REP performs the best when all the information is present. When the values of the fields of severity, priority, product, and component are left empty, the performance is similar to the result when all the information is considered. The RR@k is decreased at most 4%. However, when textual information is absent, we can observe that the RR@k decreased at most by 21%. It indicates that textual information plays a more important role than categorical information in DBRD.

Implications. Based on Figure 5, when k = 5, the best-performing approach can reach RR@k = 0.4–0.6. It indicates that a model can successfully recommend the duplicate BR in the first five positions in 40%–60% of the cases. In other words, in the rest 40%–60% of the cases, the model fails to recommend a duplicate in the first five positions. Among all the projects in our dataset, 2.7%–10% BRs are duplicates. A technique that has the RR@k of 0.4–0.6 means it can help eliminate at most 6% duplicate BRs of all the BRs. Thus, it saves considerable costs and human labor. Besides, considering the large number of candidate BRs, successfully recommending the correct master BR in the first five positions is a challenging task by itself.

Overall, the experiment results show that it is promising to directly adopt certain DBRD approaches designed based on Bugzilla data to other ITS data, such as REP. However, all the approaches demonstrate relatively poor performance in the VSCode dataset. Besides, we find that deep learning-based approaches are less robust than simpler approaches. (1) REP, which only includes handcrafted features and parameters, is considered as a simpler approach compared to deep learning approaches. REP demonstrated similar performance in both Kibana and Mozilla. However, these two datasets differ in three aspects, i.e., the ITSs, the number of BRs, and the duplicate BR rate: Mozilla contains categorical information while Kibana does not, and Mozilla has 11 times more BRs and nearly 4 times higher duplicate BR rate than Kibana. (2) Since SABD and Siamese Pair show better performances in Bugzilla data than in Jira and GitHub, it implies that it may not be ideal for applying the current deep learning-based DBRD techniques that are specifically designed for one ITS to detect duplicate BRs in different ITSs. (3) Eclipse, Mozilla, and VSCode are the largest projects in terms of the number of BRs and the duplicate BR rate. Deep learning-based SABD can achieve similar or better performance than REP. It infers that for deep learning-based approaches, they favor more training data than simpler approaches. (4) No one can win every battle. Although REP is the best performer in five of the six datasets, it loses to SABD in the VSCode dataset. It suggests that no one design is better than the rest all the time, the
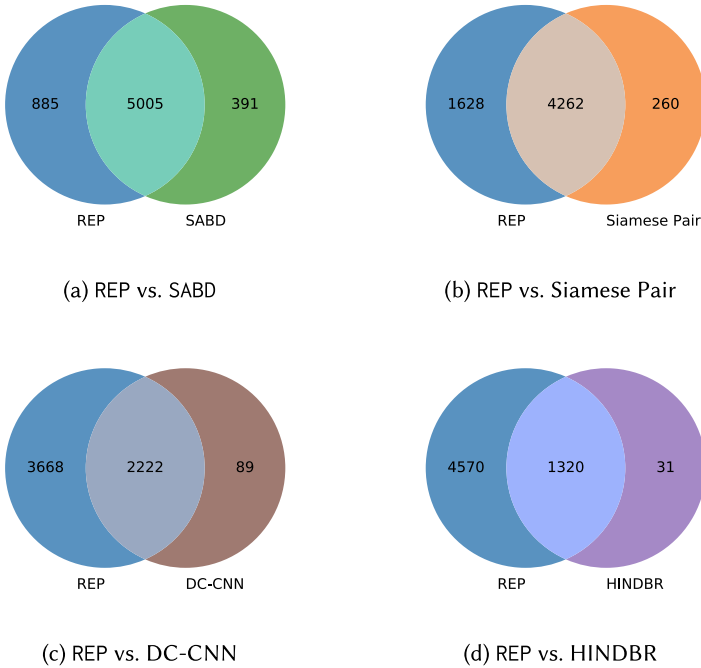
Fig. 6. REP compared to the other four approaches in terms of successful predictions.

performance of DBRD techniques could be subject to the dataset characteristics. Furthermore, we draw the Venn diagrams (Figure 6) to demonstrate that each approach can predict duplicates that the best performer cannot.

Answer 2: REP *achieves the comparative results as* SABD *on the two largest projects and it works well on smaller projects.* REP *demonstrates promising results, especially on small projects. On the other hand,* SABD *shows better performance on large projects.*

5.3 RQ3. How do the DBRD Techniques in Research Compare to those in Practice?

Full-Text Search Line FTS in Figure 5 shows the RR@k (for $k = [1 \ldots 10]$) across the six projects in comparison with the results for the research tools. We find that FTS is a competitive baseline. It can outperform HINDBR and DC-CNN for all the projects, and it can also outperform Siamese Pair on four out of six projects. In comparison with SABD (the second-best performing baseline), FTS can achieve similar performance for three out of the six projects. Still, FTS performs worse than REP on all projects. The best-performing research tool (REP) can outperform FTS by 22.1% to 62.7% in terms of RR@10.

VSCodeBot We take the intersection of VSCode BRs shown in Table 7 that are truly duplicate BRs and appear in our test set (described in Table 9). We then use this data to evaluate the performance of the various DBRD approaches. Figure 7 shows the RR@k for each DBRD approach. As VSCodeBot recommends up to five potential duplicate issues, we present RR@k, and the k ranges from 1 to 5. As shown in the figure, the two research tools, i.e., REP, SABD, achieved better performance than VSCodeBot, in terms of RR@5. Meanwhile, Siamese Pair shows a similar result
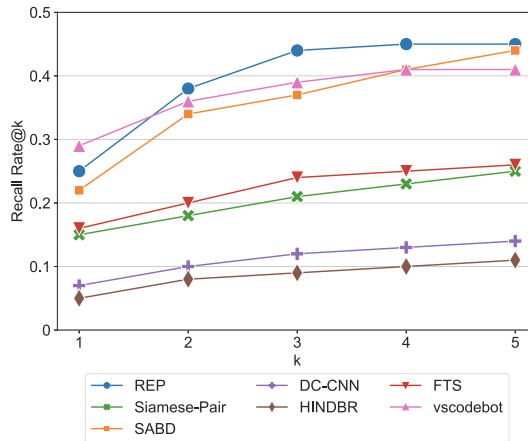
Fig. 7. Recall Rate@k comparing the tools in research and in practice on the VSCode data.

compared to VSCodeBot in terms of RR@5. FTS which is also adopted in practice also shows worse results than VSCodeBot.

Implications. Since FTS is based on exact word matching, the relatively good performance of FTS indicates that many duplicate BRs are more likely to carry the same words in BR titles. It also indicates the important role of textual information. FTS shows poor performance in the VSCode dataset, it shows that the duplicate relationship in VSCode dataset cannot be simply decided based on the words used in BR titles. However, SABD and REP achieve comparable performance as VSCodeBot on the data we investigated, which indicates it is promising to deploy research tools in practice.

> **Answer 3:** FTS *outperforms* HINDBR *and* DC-CNN *on all project data, and achieves competitive performance with* SABD *on three project data. On* VSCode *BRs,* REP *and* SABD *performed better than* VSCodeBot. *The best performing research tool (*REP*) increases the performance of* FTS *by* 46.1% *in terms of average RR@*10*, and the performance of* VSCodeBot *by* 9.8% *in terms of RR@*5*, respectively.*

6 DISCUSSION

Based on our empirical results, this section shares our insights to benefit future research on DBRD.

6.1 Age/State Bias in an Alternative ITS

As mentioned in Section 4.1 Experimental Setup, to investigate the age bias (old vs. recent data) and state bias (initial vs. latest state), we run experiments on the BRs from Bugzilla. However, it stays unknown whether age bias and state bias exist in another ITS. Here, we investigate the age and state bias on data from an alternative ITS other than Bugzilla.

Age bias in an ITS other than Bugzilla: In RQ1, the experimental results demonstrate that there is a significant difference between tools in research on the old data and recent data from Bugzilla. Besides Bugzilla, we also conducted experiments on the old data from Jira. In RQ1, for Jira, we selected Hadoop and Spark. However, since the first issue from Spark project was created on December 18, 2013. There are not enough old BRs to investigate the age bias. Note that the same reason also applies to all projects selected in our work which uses GitHub ITS. Specifically, the first issue from VScode project was created on October 14, 2015 and the first issue from Kibana

Table 11. Age and State Bias in an Alternative ITS

Bias	Data	Approach	*p*-value	|*d*|
Age	Hadoop	REP	0.65	0.13 (negligible)
		Siamese Pair	<0.001	0.98 (large)
		SABD	0.013	0.67 (large)
State	Hadoop	REP	0.199	0.35 (medium)
		Siamese Pair	0.520	0.18 (small)
		SABD	0.796	0.08 (negligible)
	Spark	REP	0.143	0.4 (medium)
		Siamese Pair	0.058	0.51 (large)
		SABD	0.043	0.54 (large)

project was created on February 6, 2013. Therefore, we are only able to conduct experiments on the old data of Hadoop, which uses Jira as an ITS. Following what we did in the RQ1, we evaluate the three tools REP, Siamese Pair, and SABD on the Hadoop old dataset (which contains BRs submitted between 2012 and 2014). Table 11 shows the statistical test results of the performance of these tools in Hadoop's old and recent data. According to the *p*-value, we can find that age bias is significant in most cases in Hadoop.

State bias in an ITS other than Bugzilla: In RQ1, the experimental results show that there is no significant difference between tools performed with the initial states and the latest states. Besides Bugzilla, we leverage the datasets shared by Montgomery et al. [37] and recover the states of issues in Jira (i.e., Hadoop and Spark) to the end of the submission day. We then perform the same experiments as RQ1 for state bias and report the results in Table 11. As Table 11 shows, the state bias is also insignificant in most cases in Hadoop and Spark, which uses Jira ITS. Note that since the issue change history in GitHub can be deleted, the saved history may be incomplete. Thus, we did not investigate the state bias in GitHub.

6.2 Performance Comparison of Tools in Research and Practice on Old Data

In RQ3, we demonstrate the performance comparison between tools in research and tools in practice. The experimental results show that REP and SABD are the best performers. FTS is better than HINDBR and DC-CNN. However, it remains unknown how different DBRD tools compare with each other in the data from old age. For the two tools in practice we evaluated, since VSCodeBot is not open-sourced, hence we can only investigate FTS. Figure 8 shows that REP and SABD are still the best performers. Siamese Pair comes in third. FTS is better than HINDBR and DC-CNN in both datasets. Even so, the performance of all the approaches dropped in Mozilla's old dataset. It suggests that when the size of a dataset becomes larger, the performances of DBRD approaches do not always improve.

6.3 Failure Analysis

To understand what are the causes of DBRD approaches that failed to detect some duplicate BRs, we investigated the three best performers, i.e., REP, SABD, and Siamese Pair. We selected the largest projects from each of the ITSs, i.e., Mozilla, Hadoop, and VSCode. We conducted the following steps to understand the causes of the DBRD failures:

(1) Firstly, we get the BRs that were not detected successfully in all five runs (out of 10 positions) by each approach in each dataset.
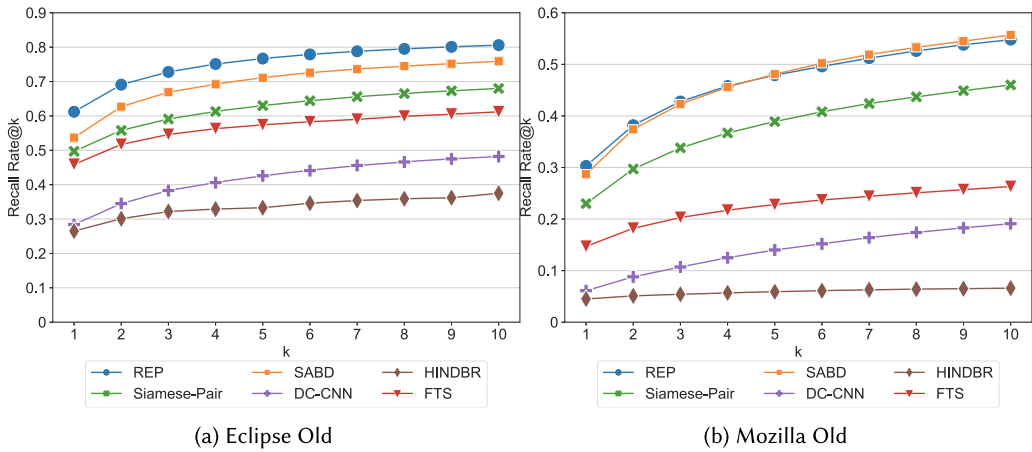(2) Secondly, we sampled 50 BRs, which failed to recommend the three approaches on the three datasets.

Fig. 8. Recall Rate@*k* in the test data of Eclipse-Old and Mozilla-Old.

We identified three causes of failed duplicate detection and describe them as follows.

(1) *Limited or Incomplete description.* When the description is short, it does not provide enough context to understand the issue. Issue reporters attach screenshots or other supporting materials in the issue so that they neglected to write a detailed description. We also found that some descriptions contain too many URLs with only limited textual information. One such example is Bug 1668483[12] from the `Mozilla` project. The description of this bug is full of long explicit URLs, which makes it hard for models to understand the real content in this issue. Furthermore, we found that issue reporters may break the BR description into multiple parts. They can write a BR description into several comments, which were not considered by our work. One such example is Bug 1641043[13] from the `Mozilla` dataset. The issue reporter actually described the issue in two consecutive parts, while only the first block is called "description" and the second block is considered as "comment". A complete version of the description may be helpful for DBRD approaches to detect duplicates.

(2) *Inability of the current approaches to understand the different ingredients in BR descriptions.* Since the current DBRD approaches only treat the textual information as unstructured, they cannot extract useful information from the description. The useful information contained in the description may describe the failure, steps to reproduce, system information, and so on. They are usually arranged in a structured way. Aside from natural language description, there can also be (1) code snippets, (2) logs, (4) backtraces inside the description. A more reasonable approach may be able to extract different types of information separately. One such example is the duplicate BR pair from the VScode dataset, i.e., issue #105446[14] and issue #110999.[15] Both issues contain the system information, steps to reproduce, and screenshots. However, since the information in the descriptions was not considered separately, it may be challenging for models to understand the failures. Other than textual information, a BR can also contain images or screen recordings in the description. However, the approaches evaluated in our work only consider the textual information in BR descriptions. Sometimes, the screenshot shows similar information. For

---

[12]https://bugzilla.mozilla.org/show_bug.cgi?id=1668483.
[13]https://bugzilla.mozilla.org/show_bug.cgi?id=1641043.
[14]https://github.com/microsoft/vscode/issues/105446.
[15]https://github.com/microsoft/vscode/issues/110999.

instance, the duplicate issues #108908[16] and #107104[17] in VScode repository. Among these two issues, issue #108908 describes the bug as "overlap" and issue #107104 describes the bug as "loads them twice", which does not look duplicate for sure. However, based on the screenshots from both issues, we can find these two issues refer to the same bug. An approach that can handle the screenshots and screen recordings would be helpful in these cases.

(3) *Different failures with the same underlying fault.* As indicated by Runeson et al. [46], there are two types of duplicates: (1) they describe the same failure; (2) they describe two different failures with the same underlying fault. We also encountered difficult cases when both BR described the issue correctly, however, they described the two different failures while the underlying fault is the same. Since the current approaches are based on the similarity of BRs, it is challenging for them to detect the second type of duplicates. One such example is the duplicate BR pair from the Hadoop dataset, i.e., HBASE-24609[18] and HBASE-24608.[19] The two BRs describe two different objects, i.e., `MetaTableAccessor` and `CatalogAccessor`. Even for developers with some experience on Hadoop projects, it may not be possible to recognize that they are duplicates.

6.4 Lessons Learned

Age bias and ITS bias should be considered for DBRD, and even for other tasks that involve BRs. We show that two kinds of bias (age and ITS) affect the performance of DBRD techniques. These biases must be considered while designing and evaluating future DBRD techniques. Furthermore, we believe that any task involving BRs, e.g., bug localization [29, 35, 39], bug severity prediction [10, 12, 34], bug triage [49, 59], and so on should also provide due consideration for these biases. When evaluating an approach, it would be better to consider the diversity6 of the ITS.

Using FTS and REP as a baseline for evaluating DBRD approaches. We observe that FTS although simple outperforms many other DBRD approaches for most projects (all except `Mozilla`). REP, although proposed a decade ago, is the overall best performer. Thus, we suggest future research include these simpler techniques as baselines. Future state-of-the-art approaches need to demonstrate superior performance over these simpler techniques.

Choose your weapon—Projects with a medium to low volume of historical BRs may not benefit from deep learning-based tools. The two best-performing tools are REP and SABD. SABD is deep learning-based, while REP is not. Comparing the performance of both tools in Figure 5, we can notice that their performance is similar for projects with the largest number of BRs (`Mozilla` and `vscode`). However, there is a clear big gap in performance for the other projects (although they still contain thousands of BRs as training data). This suggests that the applicability of deep learning-based solutions may be limited to very large ITSs with tens of thousands of BRs submitted over a few year period (considering *age bias* and data drift phenomenon [47]). For most ITSs, non-deep learning-based approaches may outperform. Note that, in our experiments, we did not use all the historical data for training since our findings in RQ1 show that there is a significant difference when applying a DBRD approach to old data and recent data. Besides, the old and recent data carry different characteristics, e.g., the number of BRs, so the predictions of the models trained in the past data may become less accurate in the recent data [62]. In addition, when training with more data, the training process takes longer and is more computationally expensive.

---

[16]https://github.com/microsoft/vscode/issues/108908.
[17]https://github.com/microsoft/vscode/issues/107104.
[18]https://issues.apache.org/jira/browse/HBASE-24609.
[19]https://issues.apache.org/jira/browse/HBASE-24608.

Table 12. Mann–Whitney-U with Cliff's Delta Effect Size $|d|$ on RQ1 with Controlling the Size of Training/Validation Pairs

Bias	Data	# Sampled pairs		Approach	p-value	$|d|$
		Training	Validation			
Age	Eclipse	3,342	108	REP	0.002	0.78 (large)
				Siamese Pair	<0.001	0.9 (large)
				SABD	0.040	0.54 (large)
	Mozilla	68,396	2,418	REP	0.003	0.76 (large)
				Siamese Pair	0.002	0.8 (large)
				SABD	0.007	0.7 (large)
ITS	Jira	626	26	REP	0.047	0.37 (medium)
				Siamese Pair	<0.001	0.81 (large)
				SABD	<0.001	0.79 (large)
	GitHub	724	28	REP	0.029	0.41 (medium)
				Siamese Pair	<0.001	0.93 (large)
				SABD	0.010	0.4775 (large)

As shown in Table 9, we identified that the number of BRs in different projects varies a lot (i.e., the number of training and validation pairs are different). The size of training data might be a confounding factor. To understand whether our findings still hold when we have the same number of training and validation pairs, we investigated the impact of the data size. We adopted the pair generation strategy used by SABD [43]. The positive pairs are all combinations of the BRs which belong to the same bucket. On the other hand, the negative pairs are randomly generated by pairing a BR from one bucket with the BR from another bucket. Since the number of positive pairs is fixed, we generated the same number of negative pairs. For each bias and each dataset, we sampled the same number of training pairs and validation pairs. For instance, when we work on age bias on the Eclipse dataset, as the old dataset has 8,668 BR pairs, while the recent dataset contains 3,342 BR pairs, we would sample the same number of 3,342 BR pairs from the old dataset (i.e., downsampling to the minority label). The number of training and validation pairs is reported in Table 12. We then conducted the same experiment for RQ1 with the sampled pairs as presented in Table 12. The p-values regard each bias are all cases. The detailed results can be found in our replication package.[20] In the table, we find that the main message of this article is still valid even if we work on the same number of training and validation pairs. Please note that the experimental data for state bias have no difference in terms of size (i.e., the numbers of BRs for before and after state are the same), so we excluded it in this additional analysis.

Future research approaches should compare with industry tools. Researchers have largely ignored the comparison of DBRD techniques with industry tools. We conducted experiments on both FTS and vscodebot. Our experiments showed that FTS and VSCodeBot can outperform many research tools. While we have highlighted the need for evaluation with industry tools in the context of DBRD, we believe, our suggestion is valid even for other software engineering tasks too. Researchers should investigate if some alternative tools are used in practice to solve the same/similar pain points and compare the performance of research tools with those "defacto" tools.

Efficiency matters for the pre-submission DBRD scenario. In the post-submission scenario, the DBRD technique has the liberty of time to predict duplicates, but it is not the case for the pre-submission scenario. The DBRD response time varies depending on the number of BRs in the ITS.

---

[20]https://github.com/soarsmu/TOSEM-DBRD.

Table 13. Averaged Seconds of Per Prediction Used by Different Approaches in
the Test Data of Eclipse, Mozilla, Hadoop, Spark, Kibana, and VSCode

ITS	Project	REP	Siamese Pair	SABD	DC-CNN	HINDBR	FTS
Bugzilla	Eclipse	0.07	0.15	1.61	2.49	0.76	0.20
	Mozilla	0.25	0.89	12.63	16.64	5.80	1.74
Jira	Hadoop	0.07	0.07	0.80	1.19	0.30	0.10
	Spark	0.13	0.05	0.69	1.02	0.28	0.08
GitHub	Kibana	0.07	0.18	1.40	2.10	0.67	0.19
	VSCode	0.08	0.70	3.53	5.56	1.92	0.40

JIT duplicate recommendation used by Bugzilla, i.e., FTS, works faster than most research tools as they only query the summary field of existing BRs. In usability engineering, a response time of over 1 second is considered to interrupt a user's flow of thought [38]. Given that users can perceive a delay difference of 100 milliseconds [15, 38], some DBRD approaches, which take over 10 seconds to predict potential duplicates, do not seem to meet the requirements. We report the seconds per prediction spent by each approach. The experiments were run on a machine with Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz (Mem: 252G) with 4 GeForce RTX 2080Ti (11G). Only one GPU was utilized when running a single deep learning model. From Table 13, we can find that the time needed for each approach to make a prediction differs in different projects. Generally speaking, for tools in research, REP, and Siamese Pair are faster than the rest approaches, while in the largest project Mozilla, the run time difference is more pronounced. REP is 3.56× faster than Siamese Pair, 50.52× faster than SABD, and 66.56× faster than DC-CNN. For the tools in practice, since VSCodeBot is not open-sourced, we cannot measure its run time on the pre-submission scenario. For FTS, we can find that it is faster than most research tools.

We suggest two potential topics for future DBRD research: (1) investigating the acceptable delay for the pre-submission DBRD scenario and (2) optimizing DBRD response time. To reduce the prediction time, future research can also consider reducing the search space. For example, instead of including all the BRs submitted within the one-year time window as candidates, further approaches can reduce the candidates first: applying a time-efficient technique, such as BM25, to filter out the BRs with a low chance of being duplicated. After that, expensive deep learning-based models can be used.

Comments should be considered. Based on our failure analysis, we found that comments in the BRs may be helpful. Especially, when the issue reporter separates the description of a BR into several parts. In the post-submission scenario, leveraging comments can provide additional information for DBRD tools to represent a BR.

Different ingredients in the description should be handled separately. Although in Bugzilla and Jira, there are dedicated fields for categorical information, we found that the description can be arranged in a structured way. It can have steps to reproduce, expected behavior, observed behavior, and so on. Issue reporters usually include images or videos inside the description. An approach that can understand different contents from the description would be beneficial. For GitHub, where the information such as system information, extensions used, and steps to reproduce, are usually included in the description, an approach that can extract all the useful information from the description would be more effective.

Other resources in the project can be considered to further improve DBRD accuracy. The current DBRD approaches are designed for detecting BRs with similar contents. If future approaches want to tackle the duplicates which have different failures, we suggest they consider other resources in the project, such as code base, to understand the relationship between different failures with the same root cause.

7 THREATS TO VALIDITY

7.1 Threats to Internal Validity

The threats to internal validity mainly relate to the correctness of our implementation. In order to mitigate these threats, we use the replication packages provided by the authors [27, 43, 50, 57]. Whenever the provided implementations are incomplete or unclear, we confirmed with the authors and perform the necessary steps. For tools in research, we have managed to replicate all the prior works on their own datasets and obtain similar results as those that are reported in the original articles. For tools in practice, we implemented the FTS based on the source code provided by `Mozilla`.[21] However, since we focus on the effectiveness of duplicate detection, we may not include all the optimizations that industry tools may apply to improve the run time efficiency.

Another threat relates to the quality of labels in our benchmark. Our benchmark extracts duplicate BR relations from three different ITSs, i.e., Bugzilla, Jira, and GitHub. For Bugzilla and Jira, BRs come with a field that allows developers to mark duplicate BRs as such explicitly. Like many prior works [32, 57, 58], our work regards the explicit labels that developers have recorded in BRs as ground truth. Considering the relation of duplicate BRs is usually recorded by the developers who are familiar with the project, we believe the chance that these BRs were linked or unlinked incorrectly is low. Different from Bugzilla and Jira, GitHub does not have a well-structured field to maintain duplicate references, so we implemented a regular expression to extract the duplicate issues based on GitHub's marking duplicates guideline.[22] However, the regular expression may introduce false positives (i.e., a comment seems like a duplicate issue reference, but actually not a duplicate). To mitigate the threat, we sampled 384 issues from each `GitHub` project (i.e., `Kibana` and `VSCode`), and two of the authors manually investigated the correctness of duplicate labels. We conclude that our regular expression is trustworthy as it showed 99.2% and 94.5% accuracy in the manual inspection for `Kibana` and `VSCode`, respectively. Again, the reason that we did not check the false negatives for GitHub data is that almost in all the ITSs, it is unlikely to have complete duplicate BR datasets, as developers may forget to mark the duplicate BRs. According to these manual investigation results, the number of mislabels on the duplicate BRs is low and they are not biased toward a specific baseline technique. In other words, it is less likely to affect our ability to perform a fair comparison. Thus, we believe this threat is minimal. Another threat exists in evaluating deep learning models. Due to the nature of deep learning-based approaches, the results can vary among different executions. We also acknowledge that the hyper-parameters used in our work may not be optimal. Their performance could possibly be better if one devotes more time to tuning the parameters.

Additionally, the tools used in practice may be optimized and updated timely, but the tools in research are not. This can be another threat to the results of our experiments, especially for RQ3. To examine this potential threat, we have checked the commit history of the FTS.[23] We found that the latest commit related to the duplicate suggestion function was made on February 11, 2014. It shows that the FTS method is not updated timely. However, since the VSCodeBot is not open-sourced, we acknowledge that it is possible that the underlying model or algorithm used by VSCodeBot is updated timely. Even so, based on the experimental results in Figure 7, VSCodeBot did not perform better than SABD and REP. Thus, we believe the effect of timely updates and optimization on the experimental results in RQ3 is minimal.

---

[21] https://github.com/bugzilla/bugzilla/blob/230d73a11989a46b0a0d3f271a1c4a260f371bd7/Bugzilla/Bug.pm.
[22] https://docs.github.com/en/issues/tracking-your-work-with-issues/marking-issues-or-pull-requests-as-a-duplicate.
[23] https://github.com/bugzilla/bugzilla/commits/5.2/Bugzilla/Bug.pm.

7.2 Threats to External Validity

The threats to external validity mainly relate to the generalizability of our findings. In order to reduce these threats, we take into consideration three popular ITSs, while prior works generally only evaluate the proposed approaches using BRs from one ITS, i.e., Bugzilla. For each ITS, we also choose two large and well-maintained projects/repositories.

Another potential threat to external validity is that our work only considers two non-deep learning-based approaches, and the finding may not be generalizable to other non-deep learning-based approaches. We found that most non-deep learning-based approaches were developed around a decade ago. Particularly, in the last five years, only very few non-deep learning-based techniques have been proposed and considered as baselines in the literature. Among the non-deep learning-based approaches and baselines, only REP is open-sourced.

7.3 Threats to Construct Validity

The threats to construct validity mainly lie in the evaluation metrics we use in our work. To diminish these threats, we follow prior DBRD works [28, 43, 50, 61] and use the RR@k.

8 RELATED WORK

8.1 DBRD Techniques and Practitioners' Perception

Many studies have developed DBRD techniques in the past decade. The state-of-the-art approaches and popular ones have been described in Section 2.3. Here, we introduce three classic works and a recent study assessing practitioners' perceptions about DBRD.

One of the pioneer studies in DBRD is by Runeson et al. [46]. They extracted textual fields in a BR (summary and description), and converted a BR into a vector of weights following the standard **Vector Space Model** (**VSM**) [48]. Duplicates are then identified by comparing the vector representation of an incoming BR to those of existing ones wrt. three well-known similarity metrics: Cosine, Jaccard, and Dice [48]. Wang et al. [53] extended Runeson et al.'s work by considering both natural language text and execution information (e.g., stack traces). They have shown that the consideration of execution information is beneficial but not many BRs contain such execution information. Sun et al. [51] trained a discriminative model via a **Support Vector Machine** (**SVM**) to classify whether two BRs are duplicates of one other with a probability. Based on this probability score, they retrieve and rank candidate duplicate BRs.

Zou et al. [63] surveyed 327 practitioners from diverse backgrounds to investigate practitioners' perceptions of DBRD and other automated techniques supporting ITS. They find that DBRD is among the top-3 techniques deemed to be the most valued and find that respondents appreciate these techniques as they can save developers' time, save reporters' time, provide hints for bug fixing, and so on.

8.2 Evaluation of DBRD Techniques

There are several studies on evaluating DBRD techniques [40, 52]. Rakha et al. [40] studied the differences between duplicate BRs before and after the introduction of JIT duplicate BR recommendation (JIT feature) in Bugzilla (in 2011). They found that duplicate BRs after 2011 (2012–2015) are less textually similar, have a greater identification delay, and require more discussion to be retrieved as BRs than duplicates before 2011. Their study also demonstrates that when evaluating the data after 2011, the experimental results of prior research would vary. These findings motivated us to experiment on BRs submitted after 2011. Based on their findings, we build the old and recent data after 2011. Our work differs from their work as we investigate the two 3-year time

window data after the JIT feature. We investigate not the impact of JIT DBRD feature introduction in Bugzilla, but rather *age bias*.

Tu et al. [52] also highlighted the fact that the BR attributes, i.e., field values, change over time. They raise a concern that several DBRD techniques use data from the future while training their models. Their study can be seen dealing with our *state* bias. They just investigate REP [50]'s accuracy from BRs on Bugzilla ITS of Mozilla and Eclipse. In this work, we investigated diverse factors that could affect the performance of DBRD techniques, and not only the *state* bias. Furthermore, we run a statistical test on the accuracy difference between using the initial state and the latest state and show that state bias does not have a statistically significant impact.

9 CONCLUSION AND FUTURE WORK

In this work, we evaluated DBRD techniques both in research and practice. We analyzed the factors affecting DBRD performance. We showed that on recent data, DBRD approaches demonstrated significantly different performance when compared to their performance on old data. Clearly, a DBRD technique that works well on data from many years ago but no longer so on recent data, should not be of much use to developers today. Therefore, future research should use a recent data benchmark for evaluation. We investigated three ITSs and two projects from each of them. We propose that future research should consider GitHub for DBRD since the existing approaches do not perform as well on GitHub as they do on Bugzilla and Jira. Taking a step ahead, we also compared the industry tools like Bugzilla's FTS and VSCodeBot. We observe that although some tools proposed in research work perform better, FTS can serve as a strong baseline for comparison. Furthermore, a DBRD technique, REP, proposed in 2011, outperforms advanced deep learning-based techniques that are recently proposed, and should also be used as a strong baseline.

Our work opens up exciting opportunities in DBRD. We would like to broaden the ITSs and work with other ITSs such as the Android Issue Tracker. In future work, we also plan to dig into the rich information available in GitHub issues, e.g., screenshots. Leveraging more information other than the pure text should help boost the accuracy of DBRD techniques. Even though several DBRD techniques are available, they make no guarantees on response time (prediction time). Developing a DBRD approach that works in real-time is another direction we wish to explore in future work.

REFERENCES

[1] 2022. GraphQL. Retrieved from https://docs.github.com/en/graphql. (Accessed on 02/10/2022).
[2] 2022. hadoop. Retrieved from https://issues.apache.org/jira/projects/HADOOP/issues. (Accessed on 02/10/2022).
[3] 2022. Jira. Retrieved from https://www.atlassian.com/software/jira. (Accessed on 02/10/2022).
[4] 2022. kibana. Retrieved from https://github.com/elastic/kibana. (Accessed on 02/10/2022).
[5] 2022. spark. Retrieved from https://issues.apache.org/jira/projects/SPARK/issues. (Accessed on 02/10/2022).
[6] 2022. vscode repository on GitHub. Retrieved from https://github.com/microsoft/vscode. (Accessed on 02/10/2022).
[7] 2022. vscodebot on GitHub. Retrieved from https://github.com/apps/vscodebot. (Accessed on 02/10/2022).
[8] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 183–192.
[9] Mehdi Amoui, Nilam Kaushik, Abraham Al-Dabbagh, Ladan Tahvildari, Shimin Li, and Weining Liu. 2013. Search-based duplicate defect detection: An industrial experience. In *Proceedings of the 2013 10th Working Conference on Mining Software Repositories*. IEEE, 173–182.
[10] Jude Arokiam and Jeremy S. Bradbury. 2020. Automatically predicting bug severity early in the development process. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*. 17–20.
[11] Atlassian. 2022. Open Source Project License Request | Atlassian. Retrieved from https://www.atlassian.com/software/views/open-source-license-request. (Accessed on 02/10/2022).
[12] Aladdin Baarah, Ahmad Aloqaily, Zaher Salah, Mannam Zamzeer, and Mohammad Sallam. 2019. Machine learning approaches for predicting the severity level of software bug reports in closed source projects. *International Journal of Advanced Computer Science and Applications* 10, 8 (2019), 10.14569.

[13] Victor R. Basili. 2007. The role of controlled experiments in software engineering research. In *Proceedings of the Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer, 33–37.

[14] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Duplicate bug reports considered harmful… really? In *Proceedings of the 2008 IEEE International Conference on Software Maintenance*. IEEE, 337–345.

[15] Jake D. Brutlag, Hilary Hutchinson, and Maria Stone. 2008. User preference and search engine latency. (2008).

[16] Bugzilla. 2022. Eclipse. Retrieved from https://bugs.eclipse.org/bugs/. (Accessed on 02/10/2022).

[17] Bugzilla. 2022. Mozilla. Retrieved from https://bugzilla.mozilla.org/home. (Accessed on 02/10/2022).

[18] Jayati Deshmukh, K. M. Annervaz, Sanjay Podder, Shubhashis Sengupta, and Neville Dubash. 2017. Towards accurate duplicate bug retrieval using deep learning techniques. In *Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 115–124.

[19] Bugzilla Wiki FAQ. 2022. How to Mark a Bug Report as a Duplicate? Retrieved from https://wiki.documentfoundation.org/QA/Bugzilla/FAQ#How_to_Mark_a_Bug_Report_as_a_Duplicate. (Accessed on 02/10/2022).

[20] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1797–1806.

[21] Wei Fu and Tim Menzies. 2017. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 49–60.

[22] Vahid Garousi, Michael Felderer, Mika V. Mäntylä, and Austen Rainer. 2020. *Benefitting from the Grey Literature in Software Engineering Research*. Springer International Publishing. Cham, 385–413. https://doi.org/10.1007/978-3-030-32489-6_14

[23] Edmund A. Gehan. 1965. A generalized wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika* 52, 1–2 (1965), 203–224.

[24] GitHub. Marking issues or pull requests as a duplicate - GitHub Docs. Retrieved from https://docs.github.com/en/issues/tracking-your-work-with-issues/marking-issues-or-pull-requests-as-a-duplicate. (Accessed on 02/10/2022).

[25] Mehdi Golzadeh, Alexandre Decan, Eleni Constantinou, and Tom Mens. 2021. Identifying bot activity in github pull request and issue comments. In *Proceedings of the 2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 21–25.

[26] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*. IEEE Press, Piscataway, NJ, 233–236. http://dl.acm.org/citation.cfm?id=2487085.2487132.

[27] Jianjun He, Ling Xu, Meng Yan, Xin Xia, and Yan Lei. 2020. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension*. 117–127.

[28] Nilam Kaushik and Ladan Tahvildari. 2012. A comparative study of the performance of IR models on duplicate bug detection. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 159–168.

[29] Misoo Kim and Eunseok Lee. 2021. Are datasets for information retrieval-based bug localization techniques trustworthy? *Empirical Software Engineering* 26, 3 (2021), 1–66.

[30] Pavneet Singh Kochhar, Xin Xia, David Lo, and Shanping Li. 2016. Practitioners' expectations on automated fault localization. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 165–176.

[31] Berfin Kucuk and Eray Tuzun. 2021. Characterizing duplicate bugs: An empirical analysis. In *Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 661–668.

[32] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Generating duplicate bug datasets. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 392–395.

[33] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. 2015. Support vector machines and word2vec for text classification with semantic features. In *Proceedings of the 2015 IEEE 14th International Conference on Cognitive Informatics and Cognitive Computing (ICCI* CC)*. IEEE, 136–140.

[34] Wenjie Liu, Shanshan Wang, Xin Chen, and He Jiang. 2018. Predicting the severity of bug reports based on feature selection. *International Journal of Software Engineering and Knowledge Engineering* 28, 04 (2018), 537–558.

[35] Pablo Loyola, Kugamoorthy Gajananan, and Fumiko Satoh. 2018. Bug localization by learning to rank and represent bug inducing changes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 657–665.

[36] Tim Menzies, Suvodeep Majumder, Nikhila Balaji, Katie Brey, and Wei Fu. 2018. 500+ times faster than deep learning:(A case study exploring faster methods for text mining stackoverflow). In *Proceedings of the 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 554–563.

[37] Lloyd Montgomery, Clara Lüders, and Walid Maalej. 2022. An alternative issue tracking dataset of public jira repositories. In *Proceedings of the 2022 IEEE/ACM 19th International Conference on Mining Software Repositories*. IEEE, 73–77.

[38] Jakob Nielsen. 1994. *Usability Engineering*. Morgan Kaufmann.

[39] Michael Pradel, Vijayaraghavan Murali, Rebecca Qian, Mateusz Machalica, Erik Meijer, and Satish Chandra. 2020. Scaffle: Bug localization on millions of files. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'20).* Association for Computing Machinery, New York, NY, 225–236. https://doi.org/10.1145/3395363.3397356

[40] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E. Hassan. 2018. Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval. *Empirical Software Engineering* 23, 5 (2018), 2597–2621.

[41] Mohamed Sami Rakha, Weiyi Shang, and Ahmed E. Hassan. 2016. Studying the needed effort for identifying duplicate issues. *Empirical Software Engineering* 21, 5 (2016), 1960–1989.

[42] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. 2004. Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management.* 42–49.

[43] Irving Muller Rodrigues, Daniel Aloise, Eraldo Rezende Fernandes, and Michel Dagenais. 2020. A soft alignment model for bug deduplication. In *Proceedings of the 17th International Conference on Mining Software Repositories.* 43–53.

[44] Jeanine Romano, Jeffrey D. Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the NSSE and other surveys. In *Proceedings of the Annual Meeting of the Florida Association of Institutional Research*, Vol. 13.

[45] Xin Rong. 2014. word2vec parameter learning explained. *CoRR* abs/1411.2738 (2014). arXiv:1411.2738. http://arxiv.org/abs/1411.2738

[46] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07).* IEEE, 499–510.

[47] Marcos Salganicoff. 1997. *Tolerating Concept and Sampling Shift in Lazy Learning Using Prediction Error Context Switching.* Springer Netherlands, Dordrecht, 133–155. DOI: https://doi.org/10.1007/978-94-017-2053-3_5

[48] Hinrich Schütze, Christopher D. Manning, and Prabhakar Raghavan. 2008. *Introduction to Information Retrieval.* Vol. 39. Cambridge University Press Cambridge.

[49] Yanqi Su, Zhenchang Xing, Xin Peng, Xin Xia, Chong Wang, Xiwei Xu, and Liming Zhu. 2021. Reducing bug triaging confusion by learning from mistakes with a bug tossing knowledge graph. In *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 191–202.

[50] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11).* IEEE, 253–262.

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

[52] Feifei Tu, Jiaxin Zhu, Qimu Zheng, and Minghui Zhou. 2018. Be careful of when: An empirical study on time-related misuse of issue tracking data. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018).* Association for Computing Machinery, New York, NY, 307–318. https://doi.org/10.1145/3236024.3236054

[53] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering.* 461–470.

[54] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. 2007. How long will it take to fix this bug? In *Proceedings of the 4th International Workshop on Mining Software Repositories. (MSR'07: ICSE Workshops 2007).* IEEE, 1–1.

[55] Eric W. Weisstein. 2004. Bonferroni correction. https://mathworld.wolfram.com/.

[56] Xin Xia, David Lo, Ming Wen, Emad Shihab, and Bo Zhou. 2014. An empirical study of bug report field reassignment. In *Proceedings of the 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering. (CSMR-WCRE).* IEEE, 174–183.

[57] Guanping Xiao, Xiaoting Du, Yulei Sui, and Tao Yue. 2020. HINDBR: Heterogeneous information network based duplicate bug report prediction. In *Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 195–206.

[58] Qi Xie, Zhiyuan Wen, Jieming Zhu, Cuiyun Gao, and Zibin Zheng. 2018. Detecting duplicate bug reports with convolutional neural networks. In *Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC).* IEEE, 416–425.

[59] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. 2010. Automatic bug triage using semi-supervised text classification. In *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE.2010).* Redwood City, San Francisco Bay, CA, 209–214.

[60] Zhengran Zeng, Yuqun Zhang, Haotian Zhang, and Lingming Zhang. 2021. Deep just-in-time defect prediction: How far are we? In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 427–438.

[61] Jian Zhou and Hongyu Zhang. 2012. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. 852–861.

[62] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. 2016. An overview of concept drift applications. Springer International Publishing, Cham, 91–114. https://doi.org/10.1007/978-3-319-26989-4_4

[63] Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, and Baowen Xu. 2018. How practitioners perceive automated bug report management techniques. *IEEE Transactions on Software Engineering* 46, 8 (2018), 836–862.