International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 15, No.1 (Jan-24)

http://dx.doi.org/10.12785/ijcds/150118

BushraDBR: An Automatic Approach to Retrieving Duplicate Bug Reports

Ra'Fat Al-Msie'deen[1]

[1]*Department of Software Engineering, Faculty of IT, Mutah University, Mutah 61710, Karak, Jordan*

Received 27 Feb. 2023, Revised 3 Jan. 2024, Accepted 6 Jan. 2024, Published 15 Jan. 2024

Abstract: A Bug Tracking System (BTS), such as Bugzilla, is generally utilized to track submitted Bug Reports (BRs) for a particular software system. Duplicate Bug Report (DBR) retrieval is the process of obtaining a DBR in the BTS. This process is important to avoid needless work from engineers on DBRs. To prevent wasting engineer resources, such as effort and time, on previously submitted (or duplicate) BRs, it is essential to find and retrieve DBRs as soon as they are submitted by software users. Thus, this paper proposes an automatic approach (called BushraDBR) that aims to assist an engineer (called a triager) to retrieve DBRs and stop the duplicates before they start. Where BushraDBR stands for Bushra Duplicate Bug Reports retrieval process. Therefore, when a new BR is sent to the Bug Repository (BRE), an engineer checks whether it is a duplicate of an existing BR in BRE or not via BushraDBR approach. If it is, the engineer marks it as DBR, and the BR is excluded from consideration for any additional work; otherwise, the BR is added to the BRE. BushraDBR approach relies on Textual Similarity (TS) between the newly submitted BR and the rest of the BRs in BRE to retrieve DBRs. BushraDBR exploits unstructured data from BRs to apply Information Retrieval (IR) methods in an efficient way. BushraDBR approach uses two techniques to retrieve DBRs: Latent Semantic Indexing (LSI) and Formal Concept Analysis (FCA). The originality of BushraDBR is to stop DBRs before they occur by comparing the newly reported BR with the rest of the BRs in the BTS, thus saving time and effort during the Software Maintenance (SM) process. BushraDBR also uniquely retrieves DBR through the use of LSI and FCA techniques. BushraDBR approach had been validated and evaluated on several publicly available data sets from Bugzilla. Experiments show the ability of BushraDBR approach to retrieve DBRs in an efficient and accurate manner.

Keywords: Software engineering, Software maintenance, Duplicate bug report retrieval, Formal concept analysis, Latent semantic indexing, Bug tracking system, Bug report.

1. Introduction

A software bug is a fault in its code, made by software coders, that prevents software from running properly. Software bugs are reported as BRs to a BTS during SM. Hundreds of BRs are submitted every day for large and complex software products (*e.g.,* Mozilla and Eclipse). Duplicated BRs arise when multiple users report multiple BRs for the same software bug [1]. Because of the asynchronous nature of the BR submission process, conventional BTSs (*e.g.,* Bugzilla) cannot avoid DBRs. Thus, some BRs remain duplicates of one another in BTSs [2]. DBRs cause a main overhead in SM process since they usually cost valuable development time, effort, cost, and resources [3]. DBR detection (*aka.* BR de-duplication) is a hot topic in the software engineering field [4] [5].

Software bugs are inevitable in software products due to their complexity. One of the most significant activities during SM is bug fixing. Software failures affected many stockholders and caused financial losses. Therefore, knowing how to correctly fix as many bugs as possible is a big help in the development of software products [2]. Nowadays, BRs have been playing an essential role in bug fixing since they offer certain details (*e.g.,* bug summary and description) to help software engineers locate and fix specific defects in software code [6].

Large software products (*e.g.,* Eclipse) continuously receive several BRs, especially after main releases when users report new bugs [7]. In order to confirm and assign the reported bug to an engineer for fixing, it is important to check if this bug has been submitted before [8]. When DBRs are found, BRs are marked as a result, thus preventing possibly redundant work and the cost of bug fixing [9]. As soon as the number of BRs is huge, detecting (or retrieving) DBRs becomes a time-consuming, costly, and error-prone task. Therefore, this paper suggests an automatic approach that aims to save time, cost, and effort and increase the accuracy of the DBR detection process. *BushraDBR* is an open-source approach for engineers to help them retrieve possible DBRs before assigning them to software developers.

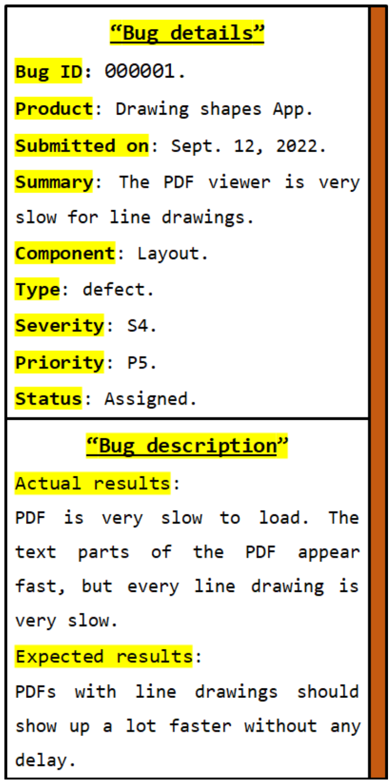
The bug reporting activity is an essential part of the SM process. Nowadays, BTS is employed by software users to maintain the track record of a software bug that is reported throughout the usage of a specific software system [10]. The key input for any BTS is BRs. BTS upholds the Master BRs set (MRs). Commonly, natural language is employed to write BRs. The same BR can be written in several ways by the software user who reports the bug. It is because the vocabulary differs across software users based on their level of technical background. The content of BR is later examined by a specialist who has a complete understanding of the software known as triager [11].

Triager has two key responsibilities. Where triager converts the wording of BR into more technical language for better comprehension by the software developers. Also, triager carries out the search activity in MRs (or BRE) for possible DBRs that have a similar (or common) signature. Furthermore, if the New BR (Nr) is non-duplicate, then it is joined to MRs; otherwise, it is deemed a DBR. On the other hand, the filtering task of DBR requires a large amount of time, manual effort, and a comprehensive understanding of BRs [12].

SM activity is referred to as the modification process of a software system after its delivery to end users in order to correct software errors (or bugs) [13]. This activity aims at improving software properties (*e.g.,* performance) [14]. Sometimes, software maintenance is important to adapt software products to a modified workplace (*i.e.,* environment). Bug triaging is a significant, tedious, and time-consuming task of SM activity [15].

DBRs are assigned to several developers for fixing the bug, which wastes developer effort and time. Thus, automating the DBR retrieval procedure is extremely valuable. It decreases the developer's time, effort, and cost. Also, the decrease in manual effort improves developers' productivity. It also reduces the cost of SM activity [16]. Each BR has two kinds of information (*i.e.,* structured and unstructured). Structured information includes specific information regarding the bug (*e.g.,* product, component, and severity of the bug). While the unstructured information includes a natural language description of the bug (*e.g.,* bug summary and description). The description of a software bug may be long or short [17]. A BR includes numerous pieces of information that are related to a specific bug or issue. Figure 1 gives an example of a BR from the *Drawing Shapes Application* (DSA) [18] [19] [20]. DSA lets the end user draw numerous types of shapes, like lines, rectangles, and ovals [21] [22].

It has been seen that often a BR reported is a duplicate, which results in large DBRs in a BTS [16]. When multiple users report BRs for the same trouble, these reports are known as DBRs. As demonstrated in existing studies, the ratio of DBRs can be up to 30% [3]. DBRs lead to a state where the same bugs are sent to several engineers who reproduce and resolve the bug for the same reason, which is considered a waste of time, effort, and cost [23]. DBRs are a big problem for Quality Assurance (QA) engineers, triagers, testers, and developers since they stimulate additional work to resolve the problem [24]. In this paper, DBR retrieval is the process of querying textually similar BRs in order to group BRs that report the same trouble (or issue) in one cluster.


"Bug details"

Bug ID: 000001.

Product: Drawing shapes App.

Submitted on: Sept. 12, 2022.

Summary: The PDF viewer is very slow for line drawings.

Component: Layout.

Type: defect.

Severity: S4.

Priority: P5.

Status: Assigned.

"Bug description"

Actual results:

PDF is very slow to load. The text parts of the PDF appear fast, but every line drawing is very slow.

Expected results:

PDFs with line drawings should show up a lot faster without any delay.

Figure 1. An example of a BR from drawing shapes application.

Table I displays a pair of DBRs from Bugzilla [25]. Please note that the two BRs shown in Table I belong to the same component, *CSS parsing and computation* [26], from the *core* product [27]. Readers can observe that they are textually similar and belong to the same product (*i.e.,* core) from Mozilla.

In this work, reports that are textually similar to each other are called DBRs. TS between BRs is good evidence that they describe the same (or similar) issue. Frequently, DBRs are reported by multiple software users. These users come from different backgrounds and use different vocabularies to describe the same (or similar) software bug.

The important role of the triager is to check the reported BRs for any possible duplicates before sending them to the BRE. A manual check of submitted BRs is a difficult task due to the huge number of BRs reported every day. On the other hand, retrieving DBR after storing it in the BRE is a tedious and expensive task for software developers. So, the

TABLE I. A pair of DBRs from Bugzilla (*i.e.,* core product).

Bug ID	A Short Summary of the Bug	Bug Description
671128 [28]	"Assertion: container didn't take ownership: 'Not Reached'"	"Assertion: container didn't take ownership: 'Not Reached', file layout/style/StyleRule.cpp, line, ... *etc.*"
803372 [29]	"Assertion: container didn't take ownership" mutating a deleted, matched CSS rule"	"Assertion: container didn't take ownership: 'Not Reached', file layout/style/StyleRule.cpp, line, ... *etc.*"

process of stopping DBRs before they start is important and very useful. Thus, current studies have suggested numerous approaches to retrieving and detecting DBRs from BRE (*cf.* Section 2). The majority of existing approaches detect DBRs within a single data set that contains all software bugs (*resp.* BRE, BTS, or MRs).

The novelty of this paper is that it proposes a textual-based approach (*i.e.,* an IR-based solution) to stopping DBRs before they start. BushraDBR prevents DBRs by continuously checking the recently reported or submitted BR against the BRs stored in the BTS. In the event that the submitted BR is textually similar to any of the BRs inside the BTS, the triager will exclude this BR from any further work and not include it in the BTS.

Figure 2 gives an illustrative example regarding BushraDBR approach workflow. BRE contains a collection of BRs (*i.e.,* BR_001, BR_002, BR_003, ..., BR_n). BushraDBR approach uses the newly submitted BR as a query (*i.e.,* Nr_Bug-ID). Then, it calculates the TS between query and BR documents based on LSI [30] [31]. Thereafter, it retrieves DBRs (if any) using FCA [32] [33]. In Figure 2, the test shows that the query document is associated with a BR document with the ID "BR_003", where the similarity score equals *0.98* (*i.e.,* the highest similarity score). Thus, in this example, BushraDBR makes a successful retrieval of DBRs.

BushraDBR approach uses LSI and FCA to retrieve DBRs. The interested reader can get further information about LSI and FCA techniques from several previous studies [34] [35] [36] [37] [38] [39]. The suggested approach was validated and evaluated on several data sets from Bugzilla (*cf.* Section 4). The results of the experiments show the ability of BushraDBR approach to retrieve DBRs when they exist.

The rest of the paper is structured as follows: A mini-systematic survey regarding DBRs detection and retrieval is presented in Section 2. Section 3 details the DBR retrieval process step by step. Experimental results are presented in Section 4. Finally, Section 5 concludes the paper and talks about the future work of BushraDBR.

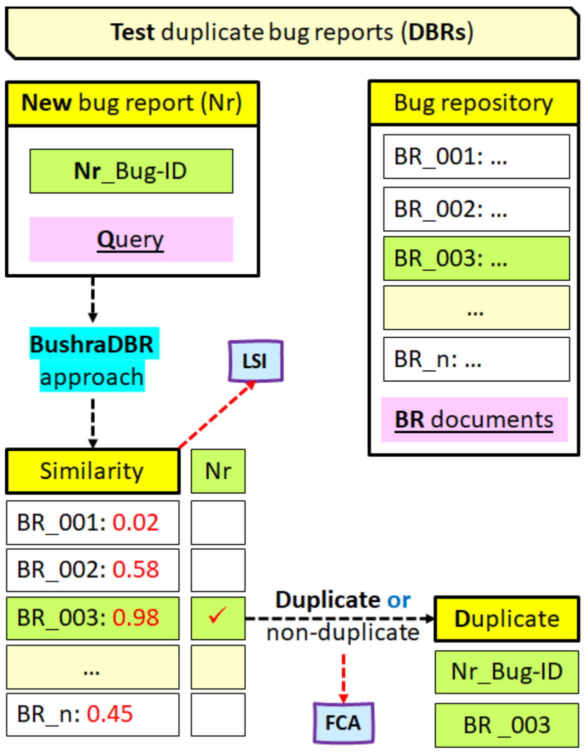


Figure 2. The workflow for retrieving DBRs via BushraDBR approach.

2. A mini-systematic survey about DBR detection and retrieval

Numerous approaches have been suggested in the literature to assist in the automatic detection of DBRs. This section offers previous approaches relevant to BushraDBR contributions.

Retrieving DBRs is a time-consuming and boring task because of the various writing styles of the huge number of submitted BRs to BTSs. Therefore, there is a need to propose BushraDBR approach in order to automate the process of DBR retrieval and avert manual effort and analysis. The main idea is to utilize BushraDBR approach for validating whether the newly submitted BR (*i.e.,* Nr) is duplicate or non-duplicate. Over the last few years, numerous studies have been suggested on the automatic retrieval and detection of DBRs. The main details of these studies are offered below:

To retrieve DBRs, Wang *et al.* [17] suggested an approach to detecting DBRs. They use natural language and execution information to obtain and determine the TS in MRs (or BRE). The experimental findings showed that the suggested approach can detect 67%-93% of DBRs in MRs of the Firefox project, compared to 43%-72% by utilizing only natural language information.

Sun *et al.* [40] proposed an approach to detecting DBRs by constructing a discriminative model that detects if two BRs are duplicates of each other or not. Their model reports a score on the possibility of X and Y being duplicates. Then, this score is utilized to retrieve related BRs from a BRE for user examination. On three BREs from Firefox, Eclipse, and OpenOffice, they have studied the feasibility of their method.

Jalbert and Weimer [3] suggested an approach that automatically classifies DBRs as they arrive at BTS. The main goal of their approach is to save developers' time. This work utilized textual semantics, graph clustering, and surface features to find DBRs. Their technique is suggested to decrease BR triage costs by discovering DBRs as they are submitted. Thus, they built a classifier for the arriving BRs to discover BR duplicates.

Sun *et al.* [41] suggested an approach based on *BM25F* to retrieve DBRs, where *BM25F* is an efficient technique for document similarity measurement. In addition to the textual information of BRs (*i.e.,* summary and description), additional information from BRs (*e.g.,* product, component, priority, *etc.*) is employed to retrieve DBRs. The authors assess their approach by generating a list of candidate DBRs for every BR indicated as *duplicate* by the bug triager, to determine whether or not the correct duplicate is a candidate. On three MRs from Mozilla, Eclipse, and OpenOffice, they applied their method. They achieved DBR detection improvement with 17 – 23% in mean average precision and 10 – 27% in recall rate@k ($1 \leq k \leq 20$).

Hindle and Onuczko [4] suggested the continuously querying method, which assists software users in finding DBRs as they type in their bug report. Their approach attempts to prevent DBRs before they start by continuously querying. Their approach has the ability to prevent duplicates before they happen in 42% of cases by building a simple IR model using TF-IDF and cosine distance to find similar BRs from their prefixed queries.

Thung *et al.* [42] have proposed a tool named *DupFinder* that finds the DBRs. DupFinder mines texts from the summary and detailed description of a new BR and BRs present in a BTS. Also, it uses the Vector Space Model (VSM) to measure the similarity of BRs and provides the software triager with a list of possible DBRs based on the TS of these BRs with the new BR.

Kukkar *et al.* [11] suggested an automatic approach for DBRs detection and classification by using the Deep Learning (DL) technique. The suggested approach has three components, which are: preprocessing, the DL model, and DBR detection and classification. Their approach utilized a Convolutional Neural Network (CNN) based DL model to obtain the appropriate feature. These appropriate features are utilized to define the similar features of BRs. Thus, the BRs similarity is computed based on these similar features. Their study calculated the similarity value of two BRs. Then, BRs are categorized as duplicate or non-duplicate based on the similarity values. The suggested approach applied to several available data sets, like Mozilla, NetBeans, and Gnome. The results are assessed using different metrics, such as an F-measure, recall@k, accuracy, precision, and recall. The accuracy rate of the suggested approach is between 85% and 99%, and its recall@k rate is between 79% and 94%, according to experimental results. The results of BurshraDBR approach are evaluated using different metrics, such as recall, precision, and F-measure. The recall rate of BurshraDBR is 100%, and its precision rate is 100%, according to experimental results, which demonstrate that BurshraDBR approach performs better than current approaches.

He *et al.* [43] have suggested a method to detect DBRs in pairs by employing CNN. They suggested creating a single representation for each pair of BRs via the Dual-Channel Matrix (DCM). DCM is fed to CNN to discover correlated semantic links between BRs. Then, the suggested method determines whether a pair of BRs is duplicate or not by using association features. They evaluate the suggested method on three data sets from Open Office, Eclipse, and NetBeans projects. Findings show that their method achieves excellent accuracy.

In [44], Runeson *et al.* have suggested a method to detect DBRs by employing Natural Language Processing (NLP). Findings demonstrate that their method can find 2/3 of DBRs by means of NLP techniques. In their work, they have described the five processing stages in NLP based on Manning and Schütze [45], which are: tokenization (or word splitting) [46], stemming (root of words) [47], English stop words removal, representation of the vector space, and, at last, calculating similarity values. From their work, BurshraDBR's approach uses some steps in the pre-processing of BRs, such as tokenization, stemming, and removing English stop words.

In the literature, there are numerous works assessing DBR detection (or retrieval) approaches [48], [49]. Rakha *et al.* [49] analyzed and assessed the changes between DBRs before and after the offering of the *Just-In-Time* (JIT) DBR recommendation feature in Bugzilla. The JIT duplicate retrieval feature was introduced in 2011 for Bugzilla 4.0 [50]. The authors have discovered that DBRs after 2011 (the period between 2012 and 2015) are less textually similar than duplicate BRs before the activation of the JIT feature in 2011.

In [51], Neysiani and Babamir proposed a study aimed at assessing the best DBR detection (or retrieval) approaches. They analyzed both *IR-based* and *Machine Learning* (ML) approaches. The study has proven that *ML-based* approaches are more effective and efficient than IR-based approaches. The research was assessed just in the Android BRE.

This section gives the most recent and relevant studies regarding BurshraDBR's contributions. Figure 3 illustrates the key elements of BurshraDBR's approach. The author summarizes the suggested approach according to the following basic elements: inputs (*i.e.,* BRs and Nr), outputs (*i.e.,* duplicate or non-duplicate BR), used techniques (*i.e.,* LSI and FCA), evaluation metrics (*e.g.,* recall), and used data sets (*e.g.,* Core, Firefox, and Eliot products from Bugzilla).

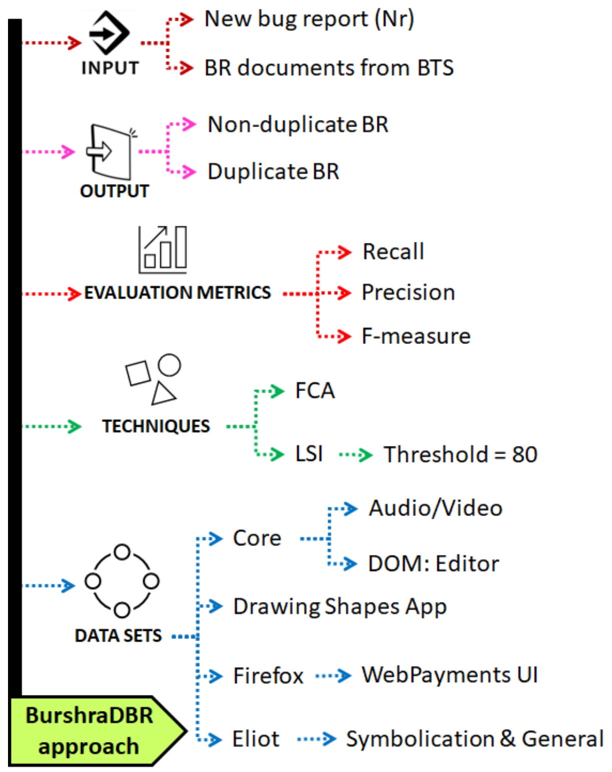


Figure 3. The *key elements* of BurshraDBR approach.

In this study, the author extracts all the essential parts for BurshraDBR from each BR, such as bug summary and description (*i.e.,* unstructured information). On the other hand, some approaches utilize structured information such as bug ID, product, and component. Table II shows the type of information that each approach utilizes in order to retrieve DBRs from MRs or BRE.

A study and comparison of current studies confirmed that there are no works in the literature that use LSI and FCA to retrieve DBRs. In this work, LSI and FCA techniques are applied in order to retrieve DBRs. BushraDBR exploits the bug's summary and description to construct a BR document. Also, BushraDBR visualizes the retrieved TS scores between BRs. Table III shows the data sets used in each study from the survey (*i.e.,* related work). The mini-systematic survey showed that the following data sets are most commonly used in research papers: Eclipse, OpenOffice, and Mozilla (*cf.* Table III).

Related work shows that there are several approaches to retrieving DBRs. The author categorizes those approaches into three categories: IR-based, ML-based, and hybrid solutions. Table IV presents the majority of existing approaches that retrieve DBRs. Table IV categorizes current studies based on the type of approach used (*i.e.,* IR, ML, or hybrid).

TABLE II. Structured and unstructured information that is leveraged by the selected approaches (*i.e.,* survey).

Reference	Structured Info				Unstructur.	
	Bug ID	Product	Component	Type	Summary	Description
Wang *et al.* [17]					×	×
Sun *et al.* [40]					×	×
Jalbert & Weimer [3]					×	×
Sun *et al.* [41]		×	×	×	×	×
Hindle & Onucz. [4]					×	×
Thung *et al.* [42]					×	×
Kukkar *et al.* [11]	×				×	×
He *et al.* [43]		×	×		×	×
Runeson *et al.* [44]		×			×	×
Rakha *et al.* [48]	×		×	×	×	×
Neysiani & Ba. [51]		×	×	×	×	×
BushraDBR	✓				✓	✓

TABLE III. Data sets that are utilized by the selected approaches (*i.e.,* survey).

	Bug report data sets									
Reference	Eclipse	Firefox	OpenOffice	Mozilla	Android	Sony Ericsson Mobile	Cyanogenmod	NetBeans	Gnome	K9Mail
Wa[17]	×	×								
Su[40]	×	×	×							
Jalb[3]				×						
Su[41]	×		×	×						
Hin[4]	×		×	×	×		×			×
Th[42]				×						
Ku[11]	×	×	×	×				×	×	
He[43]	×		×					×		
Ru[44]						×				
Ra[48]	×		×	×						
Ne[51]					×					
Bushra		✓		✓						

3. The DBR retrieval process: BushraDBR approach

The main hypothesis of BushraDBR approach is that DBRs describe the same software failure and generally use

TABLE IV. Review and classification of current studies relevant to DBR retrieval approaches.

ID	Type of approach	Approaches (or references)
01	IR-based approach	*BushraDBR*, Jalbert and Weimer [3], Sun *et al.* [41], Hindle and Onuczko [4], Thung *et al.* [42], Runeson *et al.* [44], Rakha *et al.* [49], Li *et al.* [52], Li *et al.* [53], Sureka and Jalote [54], Wang *et al.* [17], Rakha *et al.* [48], Banerjee *et al.* [55], Banerjee *et al.* [56].
02	ML-based approach	Sun *et al.* [40], Kukkar *et al.* [11], He *et al.* [43], Deshmukh *et al.* [57], Aggarwal *et al.* [58], Budhiraja *et al.* [59], Budhiraja *et al.* [60], Messaoud *et al.* [61], Wu *et al.* [62], Panichella [63], Isotani *et al.* [64], Alipour *et al.* [65].
03	Hybrid approach (IR & ML)	Bagal *et al.* [66], Tian *et al.* [67], Neysiani and Babamir [51], Nguyen *et al.* [68], Jiang *et al.* [69], Pasala *et al.* [70], Feng *et al.* [71].

similar vocabularies. Also, BushraDBR approach referred to the first reported BR for a particular bug in a software system as a master (or original) BR, while it referred to the subsequent BRs for the same bug as DBRs. Figure 4 shows two BRs that describe the *same* software failure and use *similar* vocabularies (*i.e.,* DBRs).

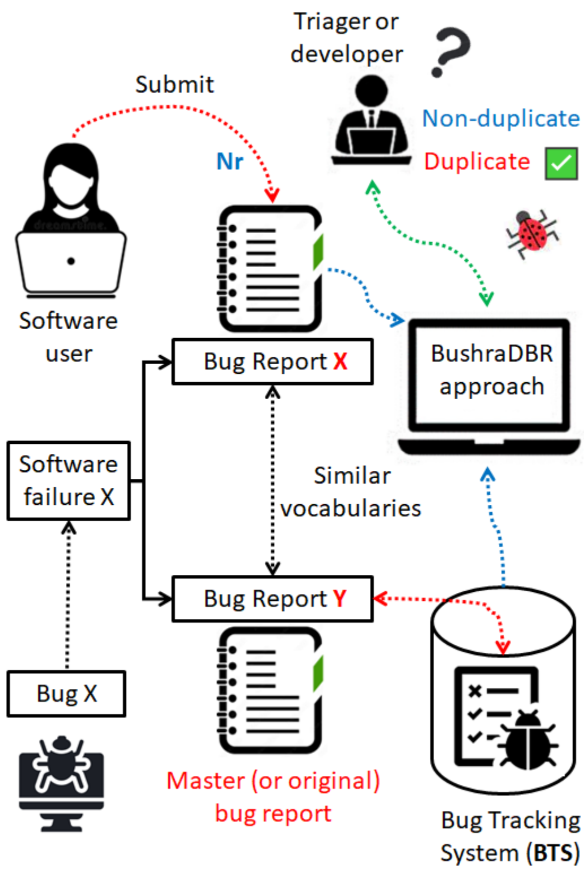


Figure 4. DBRs describe the same software failure and use similar vocabularies.

The suggested approach accepts as *inputs* a set (or subset) of BRs from BTS (or BRE) in addition to the newly submitted BR from the software user (*i.e.,* Nr). The proposed approach retrieves BRs that are textually similar to the newly reported BR if they exist as *outputs*. The contents of each BR are textual information written in natural language (*i.e.,* BR document ← summary + description). An overview of BushraDBR approach is presented in Figure 5.

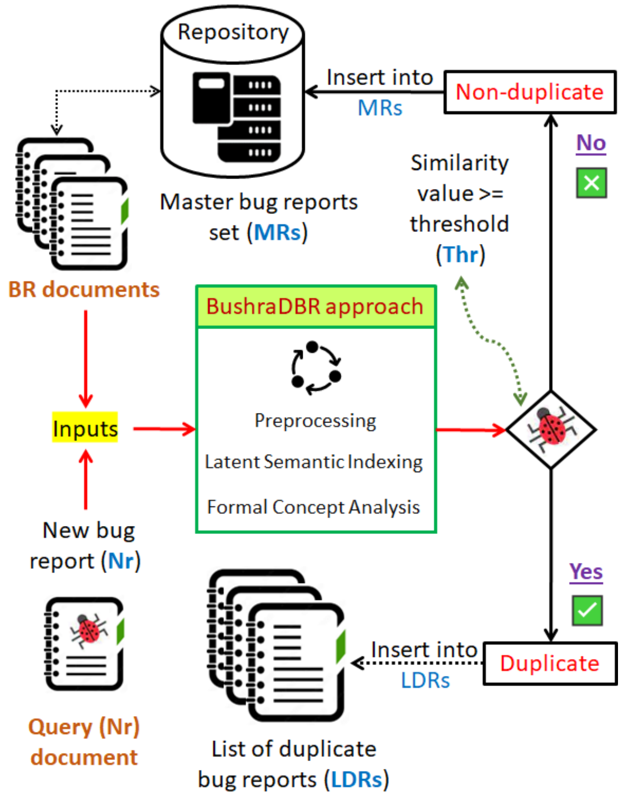


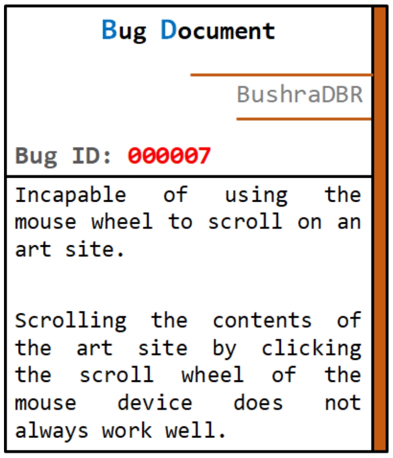
Figure 5. The DBR retrieval process - BushraDBR approach.

This study aims to prevent DBRs from entering the BTS. BushraDBR checks the BR and makes sure that it is not textually similar to another BR that was previously submitted to BTS. If a BR is similar to another BR, it is ignored, and if it is unique, it is added to the BTS. This process saves a lot of time, effort, and cost for the triager, as he only deals with unique BR (*i.e.,* non-duplicate). In some cases, BRs may define the same bug (or fault) with various words. In this circumstance, the suggested approach may fail to retrieve these DBRs because they use various vocabulary to describe the same bug. Table V displays all bug reports for DSA [72].

According to the proposed approach, BushraDBR retrieves DBRs in three steps, as clarified in the following:

A. Preprocessing of BRs

Based on the MRs and Nr, the suggested approach generates a document for each BR. This document is named by the bug ID. The contents of each document are the bug's summary and description. Figure 6 shows the bug document for the bug with ID 7 from Table V.



Bug Document

BushraDBR

Bug ID: 000007

```
Incapable of using the mouse wheel to scroll on an art site.

Scrolling the contents of the art site by clicking the scroll wheel of the mouse device does not always work well.
```

Figure 6. An example of a bug document generated by BushraDBR approach.

Algorithm 1: Preprocessing of BR documents.

Data: Master Bug Reports Set (MRs) and New Bug Report (Nr).
Result: Processed BR documents (*i.e.,* query and BR documents).

```
1  // Preprocessing of all BR documents.
2  for each bug report (BR) in MRs and Nr do
3      // Extracting BR contents.
4      Extract the bug ID, textual summary, and description of each BR
5      // Splitting BR words (or tokenization) via the camel-case splitting method [73].
6      Split the textual information of each BR into words (or tokens)
7      // Removing English stop words from each BR by using the author's list of English stop words [72].
8      Remove English stop words from each BR document
9      // Stemming BR words with WordNet [74].
10     Return each word to its base (i.e., root or stem)
11     // The query document (i.e., Nr) is named with its Bug ID.
12     Query document ← Nr document
13     // Each BR document (i.e., BR in MRs) is named with its Bug ID.
14     BR document ← BR in MRs
15 return processed documents (query and BRs)
```

The contents of each bug document are preprocessed carefully (*cf.* Algorithm 1), where the stop words are removed (*e.g.,* my, of, to, from, a, an, for, the, *etc.*), the words of each bug document are divided based on the camel-case method (*e.g.,* fillText → fill and text), and finally every word of the document is returned to its root (*e.g.,* drawing → draw).

Algorithm 1 shows the step-by-step procedure for the preprocessing of BRs using BushraDBR approach. The input instances of this algorithm are all BR documents from MRs and new BR (*i.e.,* Nr), while the output instances are the processed BR documents (*i.e.,* query and BR documents).

Natural language preprocessing is employed in the majority of current studies to process the textual information of any software artifact document. BushraDBR preprocessing steps involve extracting the text from each BR, tokenization (or splitting the words of the BR), deleting (or removing) English stop words, and, at last, word stemming. Table VI details the preprocessing steps of BushraDBR approach.

Figure 7 shows an example of a preprocessed bug document (*cf.* BR with bug ID 000007 from Table V). This BR document was generated by the BushraDBR approach. Only important words exist in this bug document. The preprocessing steps increase the accuracy of the LSI technique and eliminate noise.

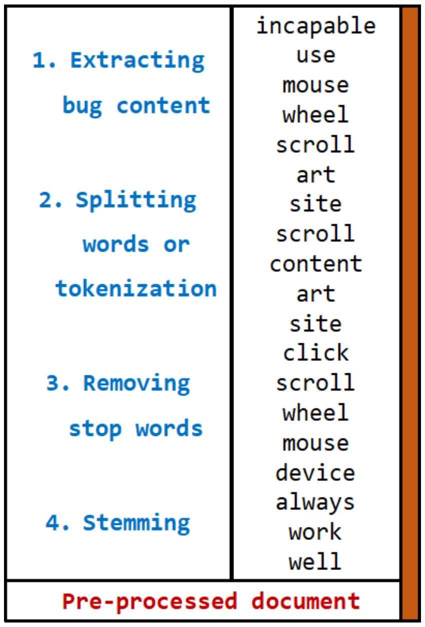


Figure 7. An example of a pre-processed bug document.

TABLE V. Bug reports from the drawing shapes application data set.

Bug ID	Short summary of the bug	Bug description
000001	The PDF viewer is very slow for line drawings.	”PDF is very slow to load. The text parts of the PDF appear fast, but every line drawing is very slow. While the expected result is that PDFs with line drawings should show up a lot faster without any delay.”
000002	Images of the myOval() and draw() functions are not displayed.	”The images in the draw() and myOval() functions are not displayed well. While the expected result is that for every condition in the draw() and myOval() functions, the images must be displayed well.”
000003	In some specific cases, the fillText function doesn't draw anything.	”In a very complex environment with a lot of sketches, the fillText method sometimes doesn't run probably and produces nothing. Thus, nothing was printed in several situations.”
000004	Drawing text in 2D is slow.	”The function of the 2D text doesn't efficiently scale to support drawing a large number of 2D texts on the screen at once. The performance of the DSA should be comparable to other drawing applications.”
000005	Problem with the drawing of 1px-wide lines.	”The inner triangle is not 1 pixel wide and is gray rather than blue. The draw() function should draw a one-pixel-wide line.”
000006	No Scrolling of the art site contents by utilizing the mouse wheel.	”Sometimes you cannot use the mouse wheel to scroll through the art site's contents. The mouse wheel is not running well on some art sites.”
000007	Incapable of using the mouse wheel to scroll on an art site.	”Scrolling the contents of the art site by clicking the scroll wheel of the mouse device does not always work well.”

TABLE VI. BushraDBR preprocessing steps, explanation, and a real-world example from the DSA data set.

Step ID	Step name	Explanation	Example
1	Extracting bug content	Obtaining the text from each BR.	Images of the myOval() and draw() functions are not displayed, ... *etc.* (*cf.* BR with bug ID 000002 from Table V)
2	Splitting words or tokenization	Eliminating digits (*e.g.,* 0, 5, 9), punctuation (*e.g.,* !, ?, :), and special characters (*e.g.,* @, $, &, #, %) from the text of each BR and splitting words based on the camel-case method [75].	images, of, the, my, oval, and, draw, functions, are, not, displayed.
3	Removing stop words	Deleting prepositions, conjunctions, and pronouns from text, such as: ”a”, ”the”, ”and”, ”of”, ”are”, ”an”, ... *etc.*	images, oval, draw, functions, displayed.
4	Stemming	Returning every word in the text to its root or stem [76].	image, oval, draw, function, display.

B. Measuring textual similarity between BRs using LSI

BushraDBR approach uses, in its core work, the LSI and FCA techniques. LSI calculates the TS scores between BRs, while FCA clusters DBRs together. Figure 8 shows the DBR retrieval process using LSI and FCA techniques.

BushraDBR bases the retrieval of DBR on the measurement of TS between BRs. This TS measure is computed using LSI. BushraDBR depends on the truth that DBRs concerned with describing a software fault are textually closer to one another than to the remainder of BRs in the data set. To calculate TS between newly submitted BR (Nr) and BRs of the data set (MRs), BushraDBR approach goes through three steps: creating the LSI corpus (*i.e.,* BushraDBR preprocessing steps; *cf.* Table VI); creating the Term-Document Matrix (TDM) (*resp.* the Term-Query Matrix (TQM)) for BRs (*resp.* Nr); and finally, creating the Cosine Similarity Matrix (CSM).

In order to employ LSI to find TS, BushraDBR creates a *corpus* that involves a set of bug documents and query(s). In BushraDBR case, each BR in BRE represents a document, while the newly submitted BR represents a query.

TDM is of size $r \times i$, where r is the number of terms used in the bug documents and i is the number of bug documents in the data set (or BRE). While a TQM is of size $r \times j$, where r is the number of terms used in newly submitted bug document and j is the number of newly submitted bug documents (*i.e.,* j is equal to 1). The terms for both matrices are various because they are obtained from different bug documents (*i.e.,* Nr and BRs from BRE). Table VII shows the TDM, while Table VIII shows the TQM.

BushraDBR uses the LSI technique to rank bug doc-

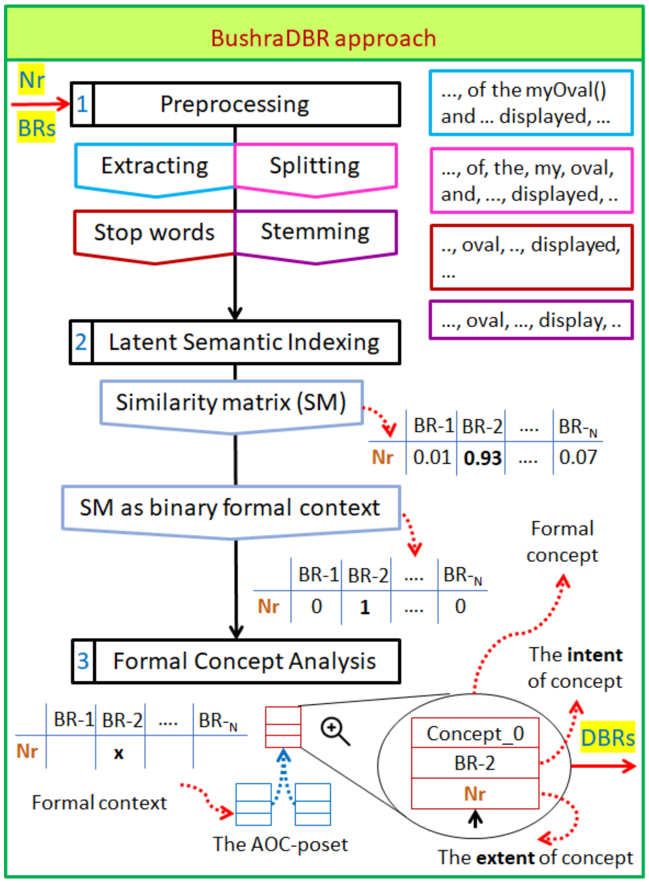


Figure 8. Retrieving DBRs using LSI and FCA techniques — BushraDBR approach.

TABLE VII. TDM for bug documents from the DSA data set (partial).

	0001	0002	0003	0004	0005	0006
method	0	0	1	0	0	0
art	0	0	0	0	0	3
site	0	0	0	0	0	3
mouse	0	0	0	0	0	3
wheel	0	0	0	0	0	3
pixel	0	0	0	0	2	0
function	0	3	1	1	1	0
content	0	0	0	0	0	2
draw	1	3	1	3	3	0
oval	0	3	0	0	0	0
image	0	3	0	0	0	0
slow	3	0	0	1	0	0
..	..	..	..	..	..	..

uments in the BTS (*i.e.,* 0001 to 0006) for the query document (*i.e.,* bug with the ID 0007), which is the new bug document (*i.e.,* Nr). BushraDBR sets the weights of terms and constructs TDM and TQM as illustrated in Tables VII and VIII.

TABLE VIII. TQM for the query document (*i.e.,* Nr) from the DSA data set (partial).

	000007
art	2
wheel	2
site	2
use	1
scroll	3
well	1
content	1
mouse	2
..	..

Similarity between bug documents (*i.e.,* Nr and BRs from BRE) is defined by a CSM (*cf.* Table IX). The columns of CSM represent vectors of BRs from BRE, while the rows of CSM represent vectors of queries (*i.e.,* Nr). Equation 1 provides a Cosine Similarity (CS) that is used to calculate TS between BRs [77] [78].

$$CS\,(BR_q, BR_j) = \frac{\overrightarrow{BR_q} \cdot \overrightarrow{BR_j}}{|\overrightarrow{BR_q}||\overrightarrow{BR_j}|} = \frac{\sum_{i=1}^{n} W_{i,q} * W_{i,j}}{\sqrt{\sum_{i=1}^{n} W_{i,q}^2}\sqrt{\sum_{i=1}^{n} W_{i,j}^2}} \quad (1)$$

TS between documents is calculated as a CS shown in Equation 1, where BR_q stands for the query vector and BR_j for the document vector. While the $W_{i,q}$ and $W_{i,j}$ measure the weights of query and document vectors, respectively. Table IX shows the calculated similarity scores between the submitted BR (*i.e.,* bug with ID 7) and the BRs from the DSA data set.

TABLE IX. Similarity scores (or CSM) of the DSA data set (*i.e.,* query and BR documents).

	000001	000002	000003	000004	000005	000006
7	-0.00043	0.01088	-0.03496	0.00317	0.00014	0.99932

Algorithm 2 shows the step-by-step procedure for measuring TS between BR documents via LSI. The input instances of this algorithm are the query document (*i.e.,* Nr), BR documents, and the chosen threshold (Thr), while the output instances are the CSM and a list of DBRs (LDRs).

Figure 9 shows the TS values between Nr and BRs from DSA as a directed graph. BushraDBR uses an external library (*i.e.,* Graphviz) to visualize the TS values between query and BR documents [79]. The directed graph in Figure 9 clearly shows the retrieved DBRs by BushraDBR approach.

C. *Retrieving DBRs using FCA*

BushraDBR employs FCA to cluster similar BRs together based on LSI results (*i.e.,* the numerical CS matrix). BushraDBR transforms the CSM resulting from the LSI

Algorithm 2: Measuring TS between BRs via LSI.

```
Data: Query document (Nr), BR documents (BRs from MRs), and Threshold (Thr).
Result: Cosine Similarity Matrix (CSM) and List of DBRs (LDRs).
1  CSM [$r_i$] [$c_j$] ← new double [$Nr_n$][$BR_n$]
2  LDRs ← ∅
3  // Compute the TS values between the query document (Nr) and all BR documents (BRs from MRs) based on the given Threshold (Thr).
4  for each query (Nr) do
5      // Compute the CS value between the query and each BR from the MRs.
6      for each BR document do
7          Compute the CS value between the query and each BR document.
8          CSM [$r_i$] [$c_j$] ← CS (query [$BR_q$], BR document [$BR_j$])
9          if $CS \geq Thr$ then
10             // The query document (Nr) is considered duplicate, and the engineer (or triager) includes it in the LDRs.
11             LDRs ← Query document (Nr)
12         else
13             // The query document (Nr) is considered non-duplicate (unique BR), and the engineer (or triager) includes it in the MRs.
14             MRs ← Query document (Nr)
15 return CSM and LDRs
```

technique into a Binary Formal Context (BFC). Table X shows an example of a Formal Context (FC). In this work, BushraDBR uses a standard threshold (Thr) for CS, which is equal to 0.80 ($CS\,\theta_{0.80}^{(BR_q,BR_j)}$). This implies that only pairs of BRs with a computed TS greater than or equal to the chosen threshold (*i.e.,* TS ≥ 0.80) are deemed DBRs. The reason (or rational explanation) behind the choice of this threshold is that DBRs generally use very similar vocabulary to describe the same bug. Also, a study of samples of DBRs available in related work (such as [5], [11], [40], and [43]) confirms the existence of common vocabulary on a large scale among DBRs.

TABLE X. A formal context of the drawing shapes application's CS matrix (*cf.* Table IX).

	0001	0002	0003	0004	0005	0006
0007	0	0	0	0	0	1

As an instance, in the BFC of Table X, the Nr with ID 0007 is associated with the BR document with ID 0006 since their TS is equal to "0.99" (*cf.* Table IX), which is higher than the chosen threshold (0.80). On the other hand, the Nr document with ID 0007 and the BR document with

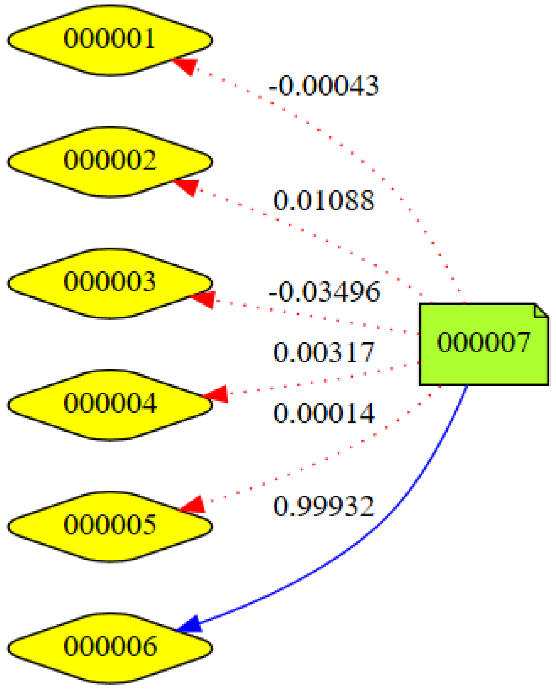


Figure 9. Textual similarity values between the query document (*i.e.,* Nr) and BRs as a directed graph.

ID 0002 are not associated since their similarity is equal to "0.01", which is less than the selected threshold. The resulting AOC-poset [80] is presented in Figure 10. This AOC-poset is comprised of a set of concepts. The extent and intent of some concepts link similar BR documents into one cluster (*cf.* Concept_0 in Figure 10).

Let's imagine that a software developer (or triager) wants to retrieve (or check) similar BRs for the newly submitted BR by a software user (*i.e.,* the bug with ID 7 from Table V). The BRs are created (or written) in English (*i.e.,* natural language). Each bug has a summary (or a short description) and a detailed description. Let's assume that the software developer is using the BushraDBR approach to check if a newly reported BR is a duplicate of existing BRs in the DSA data set or not. The software developer is checking similar BRs from BRE in order to retrieve possible DBRs regarding the recently submitted BR. Let's assume that the BR with ID 000007 is the real duplicate of the BR with ID 000006 based on the computed TS between those BRs through LSI. In this case, BushraDBR approach will classify the Nr as DBR since the suggested approach will find a lot of textually matched terms between BR documents with IDs 7 and 6. So, it is essential to consider all the textual information that is used to construct BR in BushraDBR approach.

Algorithm 3 shows the step-by-step procedure to retrieve DBRs using FCA. The input instances of this algorithm are CSM and Thr, while the output instances are AOC_{BFC}

(*i.e.,* the AOC-poset linked with *BFC*) and a list of DBRs (LDRs).

Algorithm 3: Retrieving DBRs using FCA.

```
1  // A formal context is represented by the triple F = (O, A, Br). In this triple, O is an object set (i.e., Nr) and A is an attribute set (i.e., BRs from MRs), while Br is a binary relation (i.e., Br ⊆ O × A).
2  // BFC: a formal context, where BFC = (O, A, Br).
   Data: Cosine Similarity Matrix (CSM) and Threshold (Thr).
   Result: $AOC_{BFC}$, $\leq_s$: the AOC-poset linked with BFC and List of DBRs (LDRs).
3  BFC [$O_i$] [$A_j$] ← new int [$Nr_n$][$BR_n$]
4  LDRs ← ∅
5  Thr ← 0.80
6  $AOC_{BFC} \leftarrow \emptyset$
7  // Transform the (numerical) CSM into a Binary Formal Context (BFC) based on the chosen threshold value, which is 0.80.
8  for $O_i \leftarrow 0;\ O_i < Nr_n;\ O_i \leftarrow O_i + 1$ do
9      for $A_j \leftarrow 0;\ A_j < BR_n;\ A_j \leftarrow A_j + 1$ do
10         if $CSM\ [O_i]\ [A_j] \geq Thr$ then
11             BFC [$O_i$][$A_j$] ← 1
12         else
13             BFC [$O_i$][$A_j$] ← 0
14 // Apply the FCA clustering technique via the Eclipse eRCA platform [81].
15 Generate the AOC-poset based on BFC
16 $AOC_{BFC} \leftarrow eRCA(BFC)$
17 // The resulting AOC-poset is made up of several formal concepts (e.g., concept_0, and concept_1).
18 for each concept $C \in AOC_{BFC}$ do
19     // Usually, the extent and intent of the top concept (i.e., ⊤) include DBRs (cf. Figure 10), if any.
20     if $extent(C) \neq \emptyset$ and $intent(C) \neq \emptyset$ then
21         LDRs ← extent(C)
22     else if $extent(C) \neq \emptyset$ and $intent(C) = \emptyset$ then
23         MRs ← extent(C)
24     else if $extent(C) = \emptyset$ and $intent(C) \neq \emptyset$ then
25         intent(C) ∉ LDRs
26         intent(C) ∈ MRs
27 return $AOC_{BFC}$ and LDRs
```

The efficiency of BushraDBR approach is assessed by its precision, recall, and F-Measure metrics [82]. For a given new BR document (*i.e.,* query or Nr), the recall metric is the ratio of rightly retrieved bug documents to the whole number of relevant BR documents, whereas the precision metric is the ratio of rightly retrieved BR documents to the complete number of retrieved BR documents. The F-Measure metric determines a trade-off among recall and precision metrics. Thus, F-Measure offers a high score just in conditions where both metrics (*i.e.,* precision and recall) are high. Recall, precision, and F-measure metrics are presented in Equations 2, 3, and 4. All the evaluation metrics of the suggested approach have values between zero and one.

$$Recall = \frac{relevant\ BR\ documents \bigcap retrieved\ BR\ doc.}{relevant\ BR\ documents} \quad (2)$$

$$Precision = \frac{relevant\ BR\ doc. \bigcap retrieved\ BR\ doc.}{retrieved\ BR\ documents} \quad (3)$$

$$F - Measure = 2 \times \frac{Precision \cdot Recall}{Precision + Recall} \quad (4)$$

The suggested approach relies on unstructured information (*i.e.,* bug summary and description) to retrieve DBRs from BTS. Structured information, such as component and product, does not improve the results of IR-based solutions [43].

4. Experimentation

To validate BushraDBR approach, the author conducted experiments on several data sets from Bugzilla [25]. To evaluate the results, the author used recall, precision, and F-measure metrics. To implement BushraDBR approach, a system having Windows 10 Education, Intel Core i7 processor, CPU @ 2.40GHz, and 8GB RAM is used.

In this work, the Eliot [83] open-source data set is adopted, which is produced and available at Bugzilla [84]. Eliot is the Mozilla symbolication service, and it's part of the Tecken project [85]. Tecken is a project for managing and utilizing symbols at Mozilla. The Eliot product consists of two components, which are symbolication and general. The characteristics of this data set are shown in Table XI.

The Firefox product [86] is a set of shared components utilized by the Mozilla foundation's web browser. One of these components is the WebPayments UI [87]. The WebPayments UI component of Firefox product is a user interface for the WebPayments (*i.e.,* payment request API and payment handler API). The characteristics of WebPayments UI data set are given in Table XI.

The Audio/Video component belongs to the core product [27]. This component deals with problems related to media (*i.e.,* video and audio) [88]. The data set of Audio/Video component consists of *305* BRs.

The DOM editor component belongs to the core product [27]. This component includes bugs for the DOM editor on web pages (*i.e.,* errors with editing text on the web pages). Related HTML features for this component are: ”<input type=text>, <textarea>, contenteditable, and the design-

Mode API". The data set of the DOM editor component contains *1322* BRs [89].

The most essential LSI parameter is the number of selected term-topics (*aka.* the number of topics). The BushraDBR approach cannot use a constant number of topics since it deals with different data sets. Table XI shows the selected number of topics (*i.e., K*) for each data set.

The reader can download all BRs of any data set from Bugzilla as an XML file. BushraDBR approach generates all BR documents based on this XML file. BushraDBR names each BR document by the same bug ID in the XML file.

In this study, a threshold of 0.80 (*i.e.,* Thr ← 0.80) was chosen to retrieve DBRs. This means that BRs with a TS value equal to or greater than 0.80 are considered DBRs. The reason for choosing this threshold is that BRs that use similar vocabulary with a percentage greater than or equal to 0.80 are mostly duplicates. In fact, it is not possible for a developer to find two reports that use exactly the same vocabulary (100%). But the developer can find two reports that use common vocabulary in a large ratio. BRs are usually submitted by different users to BTS. Each user uses different vocabulary to describe the submitted BR. If users submit many BRs to describe the same error, they will most likely use similar vocabulary to describe it.

Experiments show the ability of BushraDBR approach to retrieve DBRs in an efficient and accurate manner. For instance, results show that BRs with IDs *1819152* and *1819151* from the Eliot data set are DBRs. In this case, the BR with ID *1819152* (*i.e.,* Nr) is ignored and included in the LDRs. Thus, BushraDBR approach saves the engineer's time and effort, as there is no need to address this duplicate BR. Table XII shows the TS obtained from the Eliot data set.

As an illustrative example, let's say that a user has reported a BR regarding the WebPayments UI component, which belongs to the Firefox browser, by using Bugzilla BTS. The user has pressed the link to insert a new BR. A user added a summary for the BR as follows: "Need animation on the pay submit button". Then he inserted a description to this BR as follows: "As shown in UX specs, once users click on the pay submit button, we need to have an animation to show order payment sent to merchant". Finally, he pressed the submit BR button. By using BushraDBR, the approach finds that the newly added BR with ID *1510066* is very similar to the BR with ID *1490824* from the WebPayments UI data set (*cf.* Table XIII). In this case, the newly submitted BR is ignored and included in the LDRs.

A complete tutorial showing how to implement BushraDBR on the audio-video data set is available on BushraDBR web page [72]. All steps involved in the approach are fully illustrated. The purpose of this tutorial is to show how the implementation of BushraDBR works. Table XIV shows the TS of the Audio/Video data set.

The key limitation of utilizing FCA as a clustering technique in BushraDBR is that FCA works only with BFC (*i.e.,* 1 or 0). Where BushraDBR approach considers that the similarity value of 0.97 (*resp.* 0.79) is the same as 0.80 (*resp.* 0.05). Table XV shows the TS of the DOM editor data set.

Also, results show that BushraDBR approach is able to retrieve DBRs when these BRs share a common vocabulary. Otherwise, the approach will fail to retrieve DBRs. When the users use different vocabulary to describe the same bug, BushraDBR will be unable to retrieve DBRs. This means that the LSI technique may not be dependable (or should be improved with ML methods) in all cases to retrieve DBRs. For example, if we have two BRs describing the same bug and the description of the first is written as "doesn't run in dialogues", while the description of the second is written as "no longer runs as anticipated". In this case, both descriptions have the same meaning but use different vocabulary. BushraDBR will consider both reports as original BRs.

Figure 10 shows the AOC-poset for each data set. The AOC-poset of Figure 10 shows two formal concepts for each data set (*i.e.,* Concept_0 and Concept_1). In general, the top concept (*i.e.,* $\top$) contains DBRs, if any. For example, the extent and intent of Concept_0 include similar BRs (*i.e.,* DBRs). Where the extent of Concept_0 contains the query document (*i.e.,* Nr), while the intent of Concept_0 represents a similar BR from BRE.

Table XVI shows the evaluation metrics for BushraDBR's results from different data sets. Considering the recall metric, its value is 1 (or 100%) for all retrieved DBRs from the different data sets. This implies that all DBRs are correctly retrieved. Also, the precision metric value is high (100%) for all retrieved DBRs. This means that all retrieved DBRs are relevant. The F-measure value depends on the values of recall and precision metrics; thus, its value is 1 for all retrieved DBRs.

In the audio/video data set, BushraDBR retrieves the report with ID *1545237* as a duplicate of the report with ID *1545235*. The author performed a manual check of the content of each BR. The author found that the summary and description of both reports are textually similar to each other. Where the TS value between the two reports is equal to *0.84855* (*cf.* Table XIV). After conducting this manual evaluation of BushraDBR results, the evaluation metrics (*cf.* Table XVI) proved the accuracy, strength, and solidity of BushraDBR to retrieve DBRs. Thus, BushraDBR is able to prevent DBRs using LSI and FCA.

BushraDBR approach can be used only to retrieve DBRs from BTS based on the recently submitted BR. In this work, BushraDBR uses textual information to retrieve DBRs; thus, BRs vocabulary is very sensitive to BushraDBR (*i.e.,* a

TABLE XI. Characteristics of the data sets that are used in experiments.

ID	Product name	Component (*i.e.,* Data set)	Total number of BRs	Number of DBRs	*K*
1	Drawing shapes application	DSA	0007	001	006
2	Eliot	Symbolication and General	0017	001	016
3	Firefox	WebPayments UI	0126	001	125
4	Core	Audio/Video	0305	001	304
5	Core	DOM: Editor	1322	001	030

TABLE XII. Cosine similarity matrix for the Eliot data set.

	1475334	1649535	1707879	1729698	1741434	1745533	1768863	1801169	1801212
001819152	-0.00294	-0.00188	0.00907	0.01036	0.04272	-0.00197	0.00670	-0.01518	0.01662
	1811236	1811299	1812345	1814509	1815981	1815982	1819151		
001819152	-0.00454	-0.00030	0.02736	-0.01092	0.00708	-0.00400	0.99822		

TABLE XIII. Cosine similarity matrix for the WebPayments UI data set (partial).

	1427949	1427953	1432909	1432940	1490824	1432943	1432945	1432958	1435114	..
1510066	0.01822	0.00348	0.02232	0.05142	0.92465	0.00149	0.00331	0.01348	0.00261	..

TABLE XIV. Cosine similarity matrix for the Audio/Video data set (partial).

	1368902	1532646	1545235	1545847	1561960	1565493	1571258	1571583	1571912	..
1545237	0.00107	0.00186	0.84855	0.00752	0.00270	0.00391	0.00202	0.00636	0.00293	..

TABLE XV. Cosine similarity matrix for the DOM editor data set (partial).

	1005776	362694	176525	1311623	1043099	666037	1016372	229572	579763	..
001841744	0.05002	0.07664	0.96201	0.05911	0.00183	0.07517	0.01558	0.08070	0.01767	..

TABLE XVI. Evaluation metrics for BushraDBR results.

ID	Data set	Recall	Precision	F-measure
01	DSA	100%	100%	100%
02	Eliot	100%	100%	100%
03	WebPayments..	100%	100%	100%
04	Audio/Video	100%	100%	100%
05	DOM: editor	100%	100%	100%

threat to internal validity). Hence, based on the vocabulary used, BushraDBR may succeed or fail. In this study, the author validates his approach using different data sets from Bugzilla. This could pose a challenge to applying the suggested approach to another BTS in general (*i.e.,* a threat to external validity). However, the considered data sets are popular and cover different sizes. BushraDBR assumes that the user has constructed the submitted BR correctly in terms of the report summary and description. Sometimes it is possible that the BRs used are incorrectly constructed or that some parts of the BR are missing (*e.g.,* report without description), which may cause the proposed approach to be inaccurate (*i.e.,* a threat to construct validity).

Bugzilla [25] offers a feature that suggests similar BRs when a user submits a new BR (*cf.* Figure 11). The current implementation of this feature is only based on text in the summary part of BRs and does not take into account the description part of BRs. Also, the implementation does not count the frequency of words that arise in BRs. Thus, BushraDBR tool would retrieve DBRs better by using both the summary and description parts of BRs. Mozilla utilizes Bugzilla as its BTS, and Bugzilla implements an important feature to retrieve DBRs called Full-Text Search (FTS) [5]. In general, this feature relies on a BRs database (BRE) and issues SQL queries to search in the BRs database [90].

In order to retrieve DBRs via BushraDBR approach, the initial prototype was developed and accessible through the BushraDBR web page [72]. To extract the report summary and description, the author has established an XML parser for this target based on the Jdom library [91]. Jdom is an open-source Java-based solution for reading and manipulating XML files. In order to use LSI, the author has built an LSI implementation available on BushraDBR web page based on the JAMA package [92]. JAMA is a Java library for algebra, and it is able to find the Singular Value Decomposition (SVD) for specific documents belonging to a single corpus. For applying FCA, the author utilized the eRCA tool [81]. Finally, to visualize the AOC-poset (*resp.* similarity scores between BRs), BushraDBR employs the Graphviz library [79] in its implementation.

5. Conclusion and future work

This paper has introduced a novel approach called BushraDBR targeted at automatically retrieving DBRs using

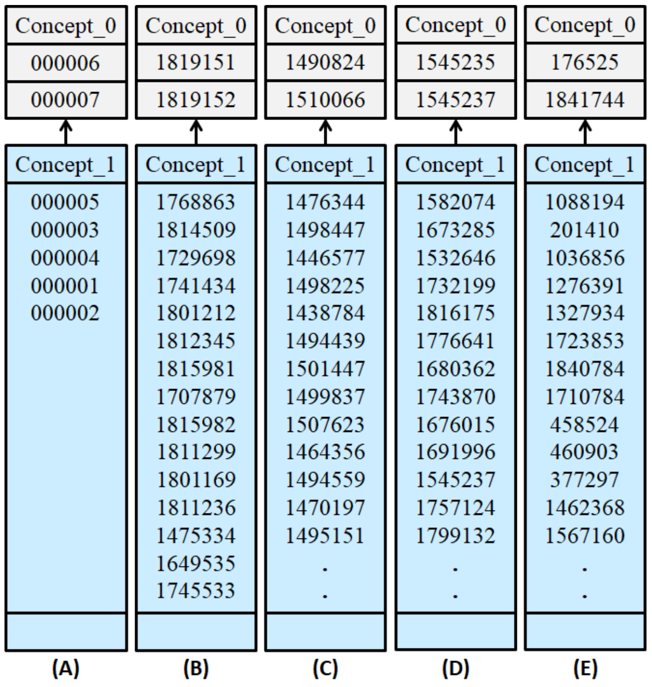
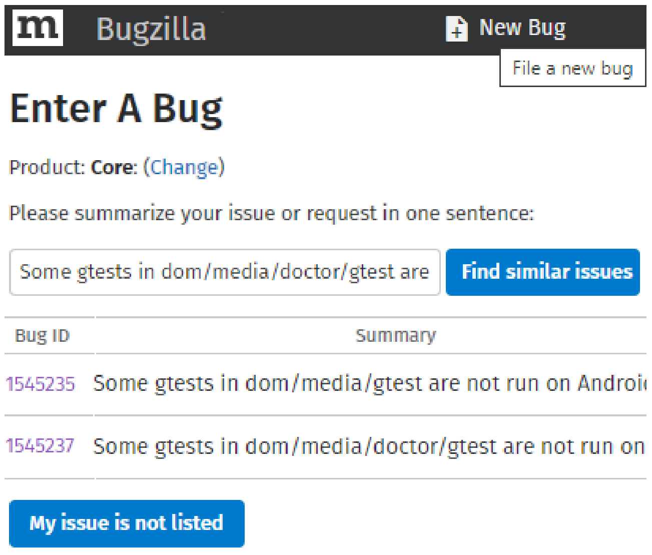


Figure 10. The AOC-poset for each data set in the experiments: (**A**) DSA, (**B**) Eliot, (**C**) WebPayments UI [partial], (**D**) Audio/Video [partial], and (**E**) DOM editor [partial].


Bugzilla

New Bug

File a new bug

Enter A Bug

Product: **Core**: (Change)

Please summarize your issue or request in one sentence:

Some gtests in dom/media/doctor/gtest are

Find similar issues

Bug ID	Summary
1545235	Some gtests in dom/media/gtest are not run on Androi
1545237	Some gtests in dom/media/doctor/gtest are not run on

My issue is not listed

Figure 11. An example of the DBR retrieval feature in Bugzilla based on the bug summary.

LSI and FCA. BushraDBR aimed to prevent developers from wasting their resources, such as effort and time, on previously submitted BRs. The novelty of BushraDBR is that it exploits textual data in BRs to apply LSI and FCA techniques in an efficient way to retrieve DBRs. BushraDBR prevents DBRs before they occur by comparing the newly reported BR with the rest of the BRs in the repository. The suggested approach had been validated and evaluated on different data sets from Bugzilla. Experiments show the capacity of BushraDBR approach to retrieve DBRs in an efficient and accurate manner. Regarding BushraDBR's future work, the author plans to extend the current approach by developing an ML-based solution to retrieve DBRs and prevent duplicates before they start. Also, he plans to compare BushraDBR (*i.e.,* an IR-based approach) with current ML-based approaches. Furthermore, additional empirical tests can be conducted to verify BushraDBR approach using open-source and industrial data sets. There is also a necessary need to conduct a *comprehensive survey* and make comparisons between all current approaches relevant to DBR retrieval.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *28th International Conference on Software Engineering (ICSE 2006), Shanghai, China, May 20-28, 2006*, L. J. Osterweil, H. D. Rombach, and M. L. Soffa, Eds. ACM, 2006, pp. 361–370. [Online]. Available: https://doi.org/10.1145/1134285.1134336

[2] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu, "How practitioners perceive automated bug report management techniques," *IEEE Trans. Software Eng.*, vol. 46, no. 8, pp. 836–862, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2018.2870414

[3] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *The 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2008, June 24-27, 2008, Anchorage, Alaska, USA, Proceedings*. IEEE Computer Society, 2008, pp. 52–61. [Online]. Available: https://doi.org/10.1109/DSN.2008.4630070

[4] A. Hindle and C. Onuczko, "Preventing duplicate bug reports by continuously querying bug reports," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 902–936, 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9643-4

[5] T. Zhang, D. Han, V. Vinayakarao, I. C. Irsan, B. Xu, F. Thung, D. Lo, and L. Jiang, "Duplicate bug report detection: How far are we?" *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, pp. 97:1–97:32, 2023. [Online]. Available: https://doi.org/10.1145/3576042

[6] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?" *IEEE Trans. Software Eng.*, vol. 36, no. 5, pp. 618–643, 2010. [Online]. Available: https://doi.org/10.1109/TSE.2010.63

[7] O. Chaparro, "Automated analysis of bug descriptions to support bug reporting and resolution," Ph.D. dissertation, The University of Texas at Dallas, 2019. [Online]. Available: https://ojcchar.github.io/publications/14-dissertation

[8] J. L. Davidson, N. Mohan, and C. Jensen, "Coping with duplicate bug reports in free/open source software projects," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2011, Pittsburgh, PA, USA, September 18-22, 2011*, G. Costagliola, A. J. Ko, A. Cypher, J. Nichols, C. Scaffidi, C. Kelleher, and B. A. Myers, Eds. IEEE, 2011, pp. 101–108. [Online]. Available: https://doi.org/10.1109/VLHCC.2011.6070386

[9] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?" in *24th IEEE International Conference on Software Maintenance (ICSM*

2008), September 28 - October 4, 2008, Beijing, China. IEEE Computer Society, 2008, pp. 337–345. [Online]. Available: https://doi.org/10.1109/ICSM.2008.4658082

[10] Palvika, Shatakshi, Y. Sharma, A. Dagur, and R. Chaturvedi, "Automated bug reporting system with keyword-driven framework," in *Soft Computing and Signal Processing*, J. Wang, G. R. M. Reddy, V. K. Prasad, and V. S. Reddy, Eds. Singapore: Springer Singapore, 2019, pp. 271–277. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-13-3393-4_28

[11] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, and K. Kwak, "Duplicate bug report detection and classification system based on deep learning technique," *IEEE Access*, vol. 8, pp. 200 749–200 763, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3033045

[12] T. S. S. Angel, G. S. Kumar, V. M. Sehgal, and G. Nayak, "Effective bug processing and tracking system," *Journal of Computational and Theoretical Nanoscience*, vol. 15, no. 8, pp. 2604–2606, 2018.

[13] IEEE/ISO/IEC-14764-2006, "ISO/IEC/IEEE international standard for software engineering - software life cycle processes - maintenance," *IEEE Computer Society*, 2006. [Online]. Available: https://standards.ieee.org/ieee/14764/3498/

[14] R. Al-Msie'deen, A. H. Blasi, and M. A. Alsuwaiket, "Constructing a software requirements specification and design for electronic it news magazine system," *International Journal of Advanced and Applied Sciences*, vol. 8, no. 11, pp. 104–118, 2021.

[15] C. Gupta, P. R. M. Inácio, and M. M. Freire, "Improving software maintenance with improved bug triaging," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 10 Part A, pp. 8757–8764, 2022. [Online]. Available: https://doi.org/10.1016/j.jksuci.2021.10.011

[16] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proceedings of the 2005 OOPSLA workshop on Eclipse Technology eXchange, ETX 2005, San Diego, California, USA, October 16-17, 2005*, M. D. Storey, M. G. Burke, L. Cheng, and A. van der Hoek, Eds. ACM, 2005, pp. 35–39. [Online]. Available: https://doi.org/10.1145/1117696.1117704

[17] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, W. Schäfer, M. B. Dwyer, and V. Gruhn, Eds. ACM, 2008, pp. 461–470. [Online]. Available: https://doi.org/10.1145/1368088.1368151

[18] R. Al-Msie'deen, H. E. Salman, A. H. Blasi, and M. A. Alsuwaiket, "Naming the identified feature implementation blocks from software source code," *Journal of Communications Software and Systems*, vol. 18, no. 2, pp. 101–110, 2022.

[19] R. A. Al-Msie'deen and A. H. Blasi, "Software evolution understanding: Automatic extraction of software identifiers map for object-oriented software systems," *Journal of Communications Software and Systems*, vol. 17, no. 1, pp. 20–28, 2021.

[20] R. Al-Msie'deen, "Requirements traceability: Recovering and visualizing traceability links between requirements and source code of object-oriented software systems," *International Journal of Computing and Digital Systems*, vol. 14, no. 1, pp. 279–295, 2023.

[21] R. Al-Msie'deen and A. Blasi, "The impact of the object-oriented software evolution on software metrics: The iris approach," *Indian Journal of Science and Technology*, vol. 11, no. 8, pp. 1–8, 2018.

[22] R. Al-Msie'deen, *Object-oriented Software Documentation*. Lap Lambert Academic Publishing, 2019.

[23] L. Kang, "Automated duplicate bug reports detection - an experiment at axis communication ab," Master's thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2017.

[24] M. S. Rakha, W. Shang, and A. E. Hassan, "Studying the needed effort for identifying duplicate issues," *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 1960–1989, 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9404-6

[25] Bugzilla-Mozilla. (2023) Bugzilla. [Online]. Available: https://bugzilla.mozilla.org/home

[26] B. Mozilla. (2023) CSS parsing and computation component. [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?product=Core&component=CSS%20Parsing%20and%20Computation&resolution=---&list_id=16624705

[27] Bugzilla.Mozilla. (2023) Core product. [Online]. Available: https://bugzilla.mozilla.org/describecomponents.cgi?product=Core

[28] Bug-671128-description. (2023) Bug 671128. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=671128

[29] Bug-803372-description. (2023) Bug 803372. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=803372

[30] H. E. Salman, A. Seriai, C. Dony, and R. Al-Msie'deen, "Recovering traceability links between feature models and source code of product variants," in *Proceedings of the VARiability for You Workshop - Variability Modeling Made Useful for Everyone, VARY '12, Innsbruck, Austria, September 30, 2012*, Ø. Haugen, J. Jézéquel, A. Wasowski, B. Møller-Pedersen, and K. Czarnecki, Eds. ACM, 2012, pp. 21–25. [Online]. Available: https://doi.org/10.1145/2425415.2425420

[31] R. Al-Msie'deen, M. Huchard, A. Seriai, C. Urtado, and S. Vauttier, "Automatic documentation of [mined] feature implementations from source code elements and use-case diagrams with the REVPLINE approach," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 10, pp. 1413–1438, 2014.

[32] R. AL-msie'deen, M. Huchard, A.-D. Seriai, C. Urtado, S. Vauttier, and A. Al-Khlifat, "Concept lattices: A representation space to structure software variability," in *2014 5th International Conference on Information and Communication Systems (ICICS)*, 2014, pp. 1–6.

[33] R. Al-Msie'deen, M. Huchard, and C. Urtado, *Reverse Engineering Feature Models*. Lap Lambert Academic Publishing, 2014.

[34] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Feature location in a collection of software product variants using formal concept analysis," in *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings*, ser. Lecture Notes in Computer Science, J. M. Favaro and M. Morisio, Eds., vol. 7925. Springer, 2013, pp. 302–307.

[35] R. Al-Msie'deen, M. Huchard, A. Seriai, C. Urtado, and S. Vauttier, "Reverse engineering feature models from software configurations using formal concept analysis," in *Proceedings of the Eleventh International Conference on Concept Lattices and*

Their Applications, Košice, Slovakia, October 7-10, 2014, ser. CEUR Workshop Proceedings, K. Bertet and S. Rudolph, Eds., vol. 1252. CEUR-WS.org, 2014, pp. 95–106. [Online]. Available: http://ceur-ws.org/Vol-1252/cla2014_submission_13.pdf

[36] R. A. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Mining features from the object-oriented source code of software variants by combining lexical and structural similarity," in *IEEE 14th International Conference on Information Reuse & Integration, IRI 2013, San Francisco, CA, USA, August 14-16, 2013*. IEEE Computer Society, 2013, pp. 586–593. [Online]. Available: https://doi.org/10.1109/IRI.2013.6642522

[37] R. Al-Msie'deen, "Reverse engineering feature models from software variants to build software product lines: REVPLINE approach," Ph.D. dissertation, Montpellier 2 University, France, 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01015102

[38] R. A. Al-Msie'deen, "Mining feature models from the object-oriented source code of a collection of software product variants," in *Doctoral Symposium of European Conference on Object-Oriented Programming (ECOOP 2013),*, Montpellier, France, July 2013, pp. 1–10.

[39] R. Al-Msie'deen, *Feature Location in a Collection of Software Product Variants*. Lap Lambert Academic Publishing, 2014.

[40] C. Sun, D. Lo, X. Wang, J. Jiang, and S. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 45–54. [Online]. Available: https://doi.org/10.1145/1806799.1806811

[41] C. Sun, D. Lo, S. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE Computer Society, 2011, pp. 253–262. [Online]. Available: https://doi.org/10.1109/ASE.2011.6100061

[42] F. Thung, P. S. Kochhar, and D. Lo, "Dupfinder: integrated tool support for duplicate bug report detection," in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, I. Crnkovic, M. Chechik, and P. Grünbacher, Eds. ACM, 2014, pp. 871–874. [Online]. Available: https://doi.org/10.1145/2642937.2648627

[43] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*. ACM, 2020, pp. 117–127. [Online]. Available: https://doi.org/10.1145/3387904.3389263

[44] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*. IEEE Computer Society, 2007, pp. 499–510. [Online]. Available: https://doi.org/10.1109/ICSE.2007.32

[45] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT Press, 2001.

[46] R. A. A. Al-Msie'deen, "Automatic labeling of the object-oriented source code: The lotus approach," *Science International-Lahore*, vol. 30, no. 1, pp. 45–48, 2018.

[47] R. A. Al-Msie'deen, "Tag clouds for software documents visualization," *International Journal on Informatics Visualization*, vol. 3, no. 4, pp. 361–364, 2019.

[48] M. S. Rakha, C. Bezemer, and A. E. Hassan, "Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports," *IEEE Trans. Software Eng.*, vol. 44, no. 12, pp. 1245–1268, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2017.2755005

[49] M. Rakha, C. Bezemer, and A. Hassan, "Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval," *Empir. Softw. Eng.*, vol. 23, no. 5, pp. 2597–2621, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9590-5

[50] Bugzilla-4.0. (2023) Release notes for bugzilla 4.0. [Online]. Available: https://www.bugzilla.org/releases/4.0/

[51] B. S. Neysiani and S. Morteza Babamir, "Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches," in *2020 6th International Conference on Web Research (ICWR)*, April 2020, pp. 288–293.

[52] Z. Li, G. Yin, Y. Yu, T. Wang, and H. Wang, "Detecting duplicate pull-requests in github," in *Proceedings of the 9th Asia-Pacific Symposium on Internetware, Internetware 2017, Shanghai, China, September 23 - 23, 2017*, H. Mei, J. Lyu, Z. Jin, and W. Zhao, Eds. ACM, 2017, pp. 20:1–20:6. [Online]. Available: https://doi.org/10.1145/3131704.3131725

[53] Z. Li, Y. Yu, T. Wang, G. Yin, X. Mao, and H. Wang, "Detecting duplicate contributions in pull-based model combining textual and change similarities," *J. Comput. Sci. Technol.*, vol. 36, no. 1, pp. 191–206, 2021. [Online]. Available: https://doi.org/10.1007/s11390-020-9935-1

[54] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *2010 Asia Pacific Software Engineering Conference*, 2010, pp. 366–374.

[55] S. Banerjee, B. Cukic, and D. A. Adjeroh, "Automated duplicate bug report classification using subsequence matching," in *14th International IEEE Symposium on High-Assurance Systems Engineering, HASE 2012, Omaha, NE, USA, October 25-27, 2012*. IEEE Computer Society, 2012, pp. 74–81. [Online]. Available: https://doi.org/10.1109/HASE.2012.38

[56] S. Banerjee, Z. A. Syed, J. Helmick, M. V. Culp, K. J. Ryan, and B. Cukic, "Automated triaging of very large bug repositories," *Inf. Softw. Technol.*, vol. 89, pp. 1–13, 2017. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.09.006

[57] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. IEEE Computer Society, 2017, pp. 115–124. [Online]. Available: https://doi.org/10.1109/ICSME.2017.69

[58] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, "Detecting duplicate bug reports with software

engineering domain knowledge," *J. Softw. Evol. Process.*, vol. 29, no. 3, 2017. [Online]. Available: https://doi.org/10.1002/smr.1821

[59] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "DWEN: deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 193–194. [Online]. Available: https://doi.org/10.1145/3183440.3195092

[60] A. Budhiraja, K. Dutta, M. Shrivastava, and R. Reddy, "Towards word embeddings for improved duplicate bug report retrieval in software repositories," in *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval, ICTIR 2018, Tianjin, China, September 14-17, 2018*, D. Song, T. Liu, L. Sun, P. Bruza, M. Melucci, F. Sebastiani, and G. H. Yang, Eds. ACM, 2018, pp. 167–170. [Online]. Available: https://doi.org/10.1145/3234944.3234949

[61] M. B. Messaoud, A. Miladi, I. Jenhani, M. W. Mkaouer, and L. Ghadhab, "Duplicate bug report detection using an attention-based neural language model," *IEEE Trans. Reliab.*, vol. 72, no. 2, pp. 846–858, 2023. [Online]. Available: https://doi.org/10.1109/TR.2022.3193645

[62] X. Wu, W. Shan, W. Zheng, Z. Chen, T. Ren, and X. Sun, "An intelligent duplicate bug report detection method based on technical term extraction," in *IEEE/ACM International Conference on Automation of Software Test, AST 2023, Melbourne, Australia, May 15-16, 2023*. IEEE, 2023, pp. 1–12. [Online]. Available: https://doi.org/10.1109/AST58925.2023.00005

[63] A. Panichella, "A systematic comparison of search algorithms for topic modelling - A study on duplicate bug report identification," in *Search-Based Software Engineering - 11th International Symposium, SSBSE 2019, Tallinn, Estonia, August 31 - September 1, 2019, Proceedings*, ser. Lecture Notes in Computer Science, S. Nejati and G. Gay, Eds., vol. 11664. Springer, 2019, pp. 11–26. [Online]. Available: https://doi.org/10.1007/978-3-030-27455-9_2

[64] H. Isotani, H. Washizaki, Y. Fukazawa, T. Nomoto, S. Ouji, and S. Saito, "Sentence embedding and fine-tuning to automatically identify duplicate bugs," *Frontiers Comput. Sci.*, vol. 4, 2022. [Online]. Available: https://doi.org/10.3389/fcomp.2022.1032452

[65] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, T. Zimmermann, M. D. Penta, and S. Kim, Eds. IEEE Computer Society, 2013, pp. 183–192. [Online]. Available: https://doi.org/10.1109/MSR.2013.6624026

[66] P. V. Bagal, S. A. JOSHI, H. D. Chien, R. R. Diez, D. C. Woo, E. R. Su, and S. Chang, "Duplicate bug report detection using machine learning algorithms and automated feedback incorporation," Patent Application Publication - United States US20 170 199 803A1, Jul. 13, 2017.

[67] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012*, T. Mens, A. Cleve, and R. Ferenc, Eds. IEEE Computer Society, 2012, pp. 385–390. [Online]. Available: https://doi.org/10.1109/CSMR.2012.48

[68] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM, 2012, pp. 70–79. [Online]. Available: https://doi.org/10.1145/2351676.2351687

[69] Y. Jiang, X. Su, C. Treude, C. Shang, and T. Wang, "Does deep learning improve the performance of duplicate bug report detection? an empirical study," *J. Syst. Softw.*, vol. 198, p. 111607, 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111607

[70] A. Pasala, S. Guha, G. Agnihotram, S. Prateek B, and S. Padmanabhuni, *An Analytics-Driven Approach to Identify Duplicate Bug Records in Large Data Repositories*. Cham: Springer International Publishing, 2016, pp. 161–187. [Online]. Available: https://doi.org/10.1007/978-3-319-31861-5_8

[71] L. Feng, L. Song, C. Sha, and X. Gong, "Practical duplicate bug reports detection in a large web-based development community," in *Web Technologies and Applications - 15th Asia-Pacific Web Conference, APWeb 2013, Sydney, Australia, April 4-6, 2013. Proceedings*, ser. Lecture Notes in Computer Science, Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, Eds., vol. 7808. Springer, 2013, pp. 709–720. [Online]. Available: https://doi.org/10.1007/978-3-642-37401-2_69

[72] R. A. A. Al-Msie'deen. (2023) BushraDBR approach. [Online]. Available: https://drive.google.com/drive/folders/1ygkB2HavOEEpLzwJUA2CE8axyLuiLfD8

[73] R. A. Al-Msie'deen and A. Blasi, "Supporting software documentation with source code summarization," *International Journal of Advanced and Applied Sciences*, vol. 6, no. 1, pp. 59–67, 2019.

[74] G. A. Miller, "WordNet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[75] R. A. Al-Msie'deen, "Softcloud: A tool for visualizing software artifacts as tag clouds," *Mutah Lil-Buhuth wad-Dirasat - Natural and Applied Sciences Series*, vol. 37, no. 2, pp. 93–116, 2022.

[76] R. Al-Msie'deen, "Tag clouds for object-oriented source code visualization," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4243–4248, 2019. [Online]. Available: https://doi.org/10.48084/etasr.2706

[77] M. W. Berry and M. Browne, *Understanding search engines - mathematical modeling and text retrieval*, ser. Software, environments, tools. SIAM, 1999.

[78] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing," in *The 25th International Conference on Software Engineering and Knowledge Engineering*. Knowledge Systems Institute Graduate School, 2013, pp. 244–249.

[79] R. Al-Msie'deen, "Visualizing object-oriented software for understanding and documentation," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 13, no. 5, pp. 18–27, 2015.

[80] R. Al-Msie'deen, A. H. Blasi, H. E. Salman, S. S. Alja'afreh, A. Abadleh, M. A. Alsuwaiket, A. Hammouri, A. J. Al_Nawaiseh,

W. Tarawneh, and S. A. Al-Showarah, "Detecting commonality and variability in use-case diagram variants," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 4, pp. 1113–1126, 2022.

[81] J.-R. Falleri and X. Dolques. (2010) Erca tool. [Online]. Available: http://code.google.com/p/erca/

[82] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Documenting the mined feature implementations from the object-oriented source code of a collection of software product variants," in *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013*, M. Reformat, Ed. Knowledge Systems Institute Graduate School, 2014, pp. 138–143.

[83] Mozilla. (2022) Eliot: Mozilla symbolication server. [Online]. Available: https://symbolication.services.mozilla.com/

[84] Bugzilla. (2023) Eliot dataset. [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?quicksearch=Eliot%20&list_id=16400624

[85] M. Foundation. (2022) Tecken: Symbols at mozilla. [Online]. Available: https://tecken.readthedocs.io/en/latest/

[86] Bugzilla.Mozilla.Org. (2023) Firefox. [Online]. Available: https://bugzilla.mozilla.org/describecomponents.cgi?product=Firefox

[87] Bugzilla. (2023) WebPayments UI. [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?product=Firefox&component=WebPayments%20UI&resolution=---

[88] Bugzilla.Mozilla-Core. (2023) Audio/video component. [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?product=Core&component=Audio%2FVideo&resolution=---&list_id=16625322

[89] Bugzilla-Core.DOM-Editor. (2023) DOM: Editor component. [Online]. Available: https://bugzilla.mozilla.org/buglist.cgi?component=DOM%3A%20Editor&list_id=16625494&product=Core&query_format=advanced&resolution=---&order=bug_id&limit=0

[90] A. A. Shamailh, R. Al-Msie'deen, and A. Alsarhan, "Comparison between the rules of data storage tools," *International Journal of Database Theory and Application*, vol. 8, no. 1, pp. 129–136, 2015. [Online]. Available: https://article.nadiapub.com/IJDTA/vol8_no1/14.pdf

[91] Jdom. (2023) The JDOM^{TM} project. [Online]. Available: http://www.jdom.org/

[92] JAMA. (2023) JAMA: A Java matrix package. [Online]. Available: https://math.nist.gov/javanumerics/jama/

Ra'Fat Al-Msie'Deen is an Associate Professor in the Software Engineering department at Mutah University since 2014. He received his PhD in Software Engineering from the Université de Montpellier, Montpellier - France, in 2014. He received his MSc in Information Technology from the University Utara Malaysia, Kedah - Malaysia, in 2009. He got his BSc in Computer Science from Al-Hussein Bin Talal University, Ma'an - Jordan, in 2007. His research interests include software engineering, requirements engineering, software product line engineering, feature identification, word clouds, and formal concept analysis. Dr. Al-Msie'Deen aimed to utilize his background and skills in the academic and professional fields to enhance students expertise in developing software systems. Contact him at rafatalmsiedeen@mutah.edu.jo. Also, you can reach him using different alternatives: author's page @ github.io, LinkedIn, ResearchGate, or Orcid.