Analysis of Duplicate Bug Report Detection Techniques

Afrina Khatun[1], Sarker Foysal Mohammud Al Gabid[2], Nazneen Akhter[3], Kazi Abu Taher[4], Tajbia Karim[5]

Abstract

The reporting of large number of duplicate bug reports has generated the need for appropriate duplicate bug report detection techniques. Researchers have developed duplicate bug report detection techniques using different approaches such as Information Retrieval, Machine Learning etc. However, due to rapid development of duplicate detection techniques, it has become difficult to compare and select an appropriate duplicate bug report detection technique. Besides, the usage of different Information Retrieval and Machine Learning techniques have made it more difficult to understand the successes, failures and future opportunities of the proposed techniques. In order to draw a clear picture of the existing techniques developed from the inception to the present, this paper presents a systematic analysis of the duplicate bug report detection techniques. The analysis has been prepared from existing techniques published in ranked conference and journals. The paper has presented insights on the type of input data set used for developing and testing the techniques, the feature selection and pre-processing strategies of bug reports and the type of algorithms and evaluation metrics used for developing the techniques. The paper lastly elaborates the findings established during the discussion of the insights, and presents a road map for future research on the uncovered areas.

Keywords: Duplicate Bug Report Detection, Feature Selection, Machine Learning, Information Retrieval, Neural Network, Deep Learning.

[1] Lecturer, Department of Information and Communication Technology, Bangladesh University of Professionals (BUP), Email: afrina.khatun@bup.edu.bd

[2] BICE Student, Department of Information and Communication Technology, Bangladesh University of Professionals (BUP), Email: soykot070917@gmail.com

[3] Lecturer, Department of Computer Science and Engineering, Bangladesh University of Professionals (BUP), Email: nazneen.akhter@bup.edu.bd

[4] Professor, Department of Information and Communication Technology, Bangladesh University of Professionals (BUP), Email: kataher@bup.edu.bd

[5] Assistant Professor, Department of Information and Communication Technology, Bangladesh University of Professionals (BUP), Email: tajbia.karim@bup.edu.bd

1. Introduction

Due to the changing nature of software systems, a bug in the source code of software is common to all software systems. These bugs are submitted in report formats for resolution. A reported bug report logically represents the information of a source code bug. However, in open source systems, large number of developers work from different locations in a single project. Therefore, it is very common that a similar bug report can be submitted by a number of developers. Besides, in open source systems, the users also have the privilege to report bugs. As a result, same bug can be reported repetitively which creates duplicate bug reports. For example 30% of Firefox's reported bug reports are duplicate of already reported bugs (Ebrahimi et al., 2019). Duplicate bug reports add cost to software development and maintenance process in terms of time and effort (Rahman et al., 2020). The identification and resolution of duplicate bugs consume the efforts of bug triggers and software developers which can be spent on unique bug reports. To reduce the cost, a number of techniques for identifying duplicate bug reports have been proposed in literature. Anvik et al. (2005) first characterized the problem of duplicate bug reports in open-source systems. The paper analyzed the bug reports from two open-source projects such as Eclipse and Mozilla Firefox. The analysis characterizes the bug reports from four different perspectives such as - the type of bug reports, the rate of bug reporting, the duration of bug resolution and the role of bug resolver.

In order to conduct the systematic analysis, the literature is searched in a step by step approach. At first, a google scholar search has been conducted using the query string "Duplicate Bug Report Detection". However, the google search technique returns all matching results that have the query string terms in any segment of the searched articles. If the google scholar search is performed by setting the option named "sort by relevance" as checked, then the false positive results occur at the end of the returned search list. Therefore, the search is performed by setting the option "sort by relevance" checked for each of the years from 2017 to 2021. For each year, the top most 100 articles have been identified. Therefore, a total of 500 articles have been collected for processing in the next phase. During the second phase, the articles which are non-English, database files, books or letter type articles are excluded. Later, the title and abstract of the full text articles are checked for relevancy with the searched query string. Out of the 500 identified articles from year 2017 to 2021, total 74 articles are found to propose different techniques of duplicate bug report identification. In the last phase, the ranking of the conferences and journals where the 74 articles are published is checked using Computing Research and Education (CORE) site (Education, 2021). Out of the 74 articles, 23 articles are published in ranked conferences and journals. Therefore, these total 23 articles are selected for the final systematic analysis.

For analyzing the existing techniques and identifying the future scope in duplicate bug detection, this paper tends to answer three Research Questions (RQs).

- RQ1 demonstrates the input data types used for duplicate bug report detection.
- RQ2 identifies the bug report properties which are used by existing duplicate bug detection approaches.
- RQ3 discusses about the algorithms which are being used for duplicate bug report detection.

The rest of the paper is organized into following sections. Section 2 shows the general model of a duplicate bug report detection technique. Section 3 discusses the existing duplicate detection techniques. Section 4 answers RQ1 by enlisting the input data sets used for developing and assessing the existing techniques. The selection and pre-processing strategies of bug report features is discussed in Section 5. This section also answers RQ2 by identifying the most used bug report features. Section 6 answers RQ3 by describing the machine learning, information retrieval and deep learning approaches used for developing the existing approaches. Finally, section 7 concludes the paper by summarizing the findings and pointing the future research scopes in the field of duplicate bug detection.

2. General Model of Duplicate Bug Report Detection Techniques

The overall duplicate bug detection techniques consist of some common sequential steps. Figure 1 represents the general model of duplicate bug report detection technique which consists of *Feature Selection*, *Bug Report Pre-processing* and *Learning Algorithm or Model*. The bug reporting systems such as Bugzilla, Jira etc. enable the developers and users to report bugs using customer friendly software interface. The tracking systems collect and maintain a variety of information about the reported bugs as bug properties (Serrano & Ciordia, 2005). These properties include - bug report id, assignee information, reporting date, component details, product details, bug severity, bug tossing and management history, bug status, bug severity, developer’s comments during bug resolution, bug summary, bug description and many more. Some secondary properties such as bug screen shots, bug stack trace, link to source code commit messages etc. are also included sometimes. Each of the property represent different information regarding the bug. The Feature Selection phase selects appropriate properties from the bug report for learning the algorithm or model in the latter phases.

The properties of bug report are written in different formats such as natural language, categorical format etc. Natural language text may contain redundant information which do not represent the actual characteristics of the reported bug. Therefore, the Pre-processing phase processes the content value of bug report properties. Tokenization, stemming, stop word removal are some of the known text pre-processing techniques used by existing techniques. Each property of the bug report represent different information of the reported bug. Finally, in the Learning Algorithm or Model a model is prepared using either machine learning, deep learning or information retrieval based techniques. The features are fed into these models identifying the duplicates. For machine or deep learning based approaches, a bug report is generally classified as duplicate or not. On the other hand, information retrieval based techniques return a ranked list of bug reports suggested to be duplicates.

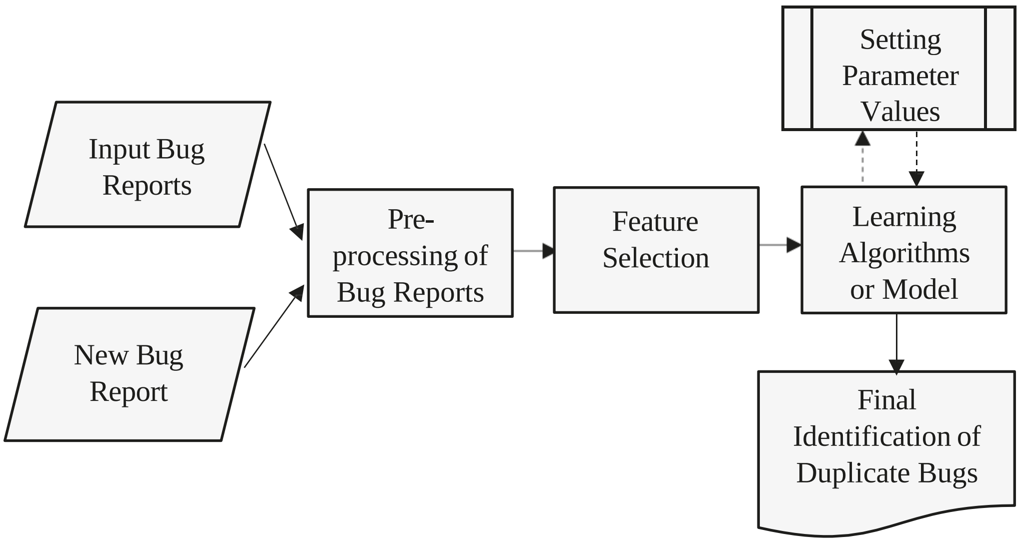

Setting Parameter Values
Input Bug Reports
Pre-processing of Bug Reports
Feature Selection
Learning Algorithms or Model
New Bug Report
Final Identification of Duplicate Bugs

Figure 1: Components of General Duplicate Bug Report Detection Techniques

3. Literature Review

With the increasing number of duplicate reports in the bug repository, the need for duplicate bug detection techniques becomes inevitable. In this essence, a number of duplicate detection approaches have been devised. The existing techniques differ from each other in using bug report properties, applying preprocessing techniques, implementing algorithms and evaluating the results. Therefore, this section describes the existing duplicate bug report detection techniques.

Jalbert and Weimer (2008) devised a duplicate bug report classification technique that classifies the bug report as duplicate as the report arrives in the system. The technique comprises of three steps which are surface (severity, OS version and number of source patch or screenshot attachment) and textual (title and description) feature extraction, text similarity measurement and graph clustering. The similarity value of textual features are calculated using vector dot product. A graph is then induced where the nodes represent bug reports and edges represent their calculated similarity value. Next, the existing reports are clustered using graph clustering algorithm. On arrival of a new report, the textual similarity is checked and the report is classified in the clusters for duplicate identification. The experiment includes a representation of a real-time bug reporting environment using 29,000 bug reports from Mozilla project. The proposed approach achieved around 52% recall rate in identifying duplicates, however the rate seems poor in comparison with other existing techniques.

Johannes and Mira (2013) proposed an approach that applies stack traces to machine learning algorithms for detecting bug report duplicates. However, the technique may fail if source code attachments are missing in bug reports. Neysiani and Babamir (2019a) also presented a duplicate detection approach which combines both IR and ML based algorithms. The paper shows that when combined the IR based technique shows similar performance as ML based techniques. Hence, ML based models provide better performance in duplicate detection than the IR based models alone.

Tian et al. (2016) developed a unified model for bug report assignee recommendation by combining developer’s activity information and bug localizing information. The proposed model extracts the title and summary text from the bug reports, and the comments and identifiers from the source code files. These extracted text then go through various text processing steps such as - tokenization, stemming and stop word removal. The model also creates a list of developer profiles which represent the list of bug reports and source code files worked on by the corresponding developers. Next, to train the assignee recommendation model sixteen (16) different features are proposed and extracted using the processed texts of bug reports and source code files. Among the sixteen features, the first five (5) features indicate the similarity between the new bug report and the source code files added, edited or deleted by a developer. In order to calculate the similarity value, cosine similarity and BM25 technique is applied. The features six (6) to ten (10) extract the similarity between the new and the previously fixed bug reports by a developer. Feature eleven (11) and twelve (12) represents the frequency and recency of a developer's bug resolution. Features 13 to 16 enables the model to incorporate the location information of bug reports and developer commits. The similarity between the new bug report's buggy code and developer's previously fixed source code is measured using the cosine similarity and BM25 technique, for forming features thirteen (13) and fourteen (14). Lastly, feature fifteen (15) and sixteen (16) checks if a developer has ever touched the source code files corresponding to the new bug report. When a new bug report arrives, the model recommends a ranked developer's list by calculating a weighted sum of k features selected from the 16 features (Lee & Lin, 2014). More features can be added, as the work of Neysiani et al. (2020) shows that consideration of product feature plays vital role. Besides, the k selected features for the three data sets show that the features from activity information play better role than location based ones.

Ebrahimi et al. (2019) devised a duplicate bug report detection technique which leverages the stack traces of bug reports. The bug report properties such as product, component and status is examined to identify the master bug report and its corresponding duplicates. Then, the identified bug reports are used to create Duplicate bug report Groups (DG) where each DG contains the stack trace of the master bug and its corresponding duplicates. In this case the bug reports having minimum four stack traces are taken into consideration. Next, the bug report of each DG is used to train a Hidden Markov Model (HMM). The stack traces of bug reports are treated as hidden state sequences and the DG groups are treated as observations in the HMM training. Baum-Welch algorithm is used to estimate the three parameters of HMM. For each DG group 60%, 10% and 20% bug reports are used for training, validation and testing of the HMM model respectively. When a new bug report arrives, the stack trace of the report is collected and matched with each of the trained HMM to identify the probability of matching score. A highest probability score represents the highest probability of being duplicate of the existing DG groups. The technique will fail for bug reports which does not contain any stack traces.

Neysiani and Babamir (2019b) presented a Longest Common Sequence (LCS) based new feature for measuring similarity between two bug reports. From each bug report identical, categorical, textual, and contextual features are elicited and converted into a sequence of features. The textual and contextual features are measured using BM25F and cosine similarity respectively. Next, the longest common sequence is calculated between the bug reports for identifying duplicates.

The proposed technique achieves a high accuracy, precision and recall rate of 96.09%, 98.43% and 97.27% respectively. The paper added a new feature with the existing ones, however, finding the appropriate feature for calculating similarity between bug reports is still an unanswered research question.

Neysiani and Babamir (2019c) proposed the individual Manhattan distance comparison method as an alternative to cosine space similarity for each subject in contextual features (Neysiani & Babamir, 2019c). The study is an extension to previous papers with using the different similarity metric. The proposed model was trained by decision tree and linear regression classifiers and achieved accuracy 96.65% and 81% respectively and recall rate 95.89% and 73.18% respectively. Wang et al. devised an duplicate detection approach which considers the natural language info and source code execution info. The approach first calculates the Natural Language based Similarities (NL-S) between the new and existing bug reports. Next, the Execution information based Similarities (E-S) are measured between the new and remaining bug reports. Finally, reports are selected using comparisons based on two heuristics, where the first is to merge the E-S and the NL-S into one joint comparison and the second is to try to identify whether the natural language or the implementation info is the main issue in perceiving the duplicate reports. The technique used TF-IDF similarity measure for checking equality which tends to achieve poor accuracy in comparison with machine learning approaches.

Poddar et al. (2019) proposed a neural architecture to identify duplicate reports by considering the latent issues of the reports. The paper applied IR based topic modeling for identifying the bug reports. However, the paper mentioned that bug report written by general users and experienced developer differ from each other in terms of technical word selection. Therefore, considering the bug reports separately based on their submitter can also affect the performance of duplicate detection. Soleimani and Morteza (2020) presented an approach to estimate the impact of typo on Duplicate Bug Report Detection (DBRD). The evaluation of the approach on the Android dataset shows that the typos improvement can increase the accuracy and recall of DBRD at most 1% in average, which is trivial. The proposed technique also used the textual, categorical, contextual and temporal features. The results indicated that removal of typos cannot improve the performance of existing techniques.

Jianjun et. al (2020) proposed a duplicate bug detection technique using Dual-Channel Convolutional neural network. First, the technique extracts structured (such as component, product, priority) and unstructured (such as summary, description) information from the bug report and converts them into a text document. Next, the text documents go through pre-processing steps which include tokenization, stemming, stop word removal and case conversion. After pre-processing, the words of the bug reports are converted into a corpus using word2vec model. As a result, each word is represented using a single dimension vector and each bug report is represented using a two dimension matrix containing the vectors. The duplicate bug report pairs are then represented by combining the single-channel bug report matrices into dual-channel bug report pairs. Next, the training phase fed the dual-channel bug report matrices into the Convolutional Neural Network (CNN) for training the model. The similarity score is compared with a threshold value to classify the bug report pairs as duplicates. The proposed technique considered only 300 words from each bug reports which may ignore important information of the remaining part of the bug reports. Besides the technique ignored the source code attachment or files while consideration of the bug report properties.

Neysiani et al. (2020) proposed an efficient feature extraction model for duplicate bug report detection. The technique starts with pre-processing the existing bug reports by null value detection, homologize bug report formats, tokenization of keywords etc. After pre-processing, the duplicate and non-duplicate bug reports are listed and inputted into the feature extraction phase. For extracting efficient features from bug reports, this phase considers four types of features which are textual, temporal, categorical and contextual. The textual feature are extracted using TFIDF, BM25F, Longest Common Sub-sequence (LCS) and aggregated functions such as maximum, minimum and average values of conventional TF and IDF values in uni-gram and bi-gram forms. The temporal features are extracted using the interval between bug report ids and opening dates. The categorical features are collected by comparing the similarity of product, component, priority, type and version information of bug reports. Contextual features are elicited by calculating the cosine similarity between contents of bug reports. Next, the efficiency of each extracted feature is checked using a Efficiency Detector Value (EDV). The EDV value is calculated using a new heuristic approach that takes the weighted average of the normalized information gain ratio, Gini index, chi-square, Principal Component Analysis and correlation of each feature. Using the extracted features a duplicate bug report detector model is generated. Although the model takes only 5 minutes to predict if a bug report is duplicate, the model compares each new bug with every existing bug report which requires huge computation.

The paper also mentioned that the contextual features alone cannot predict the duplicate bug report effectively.

Alkhazi et al. (2020) proposed an extended version of Tian et. al by adding four more features which can be used in a ranking model. The proposed technique adds four features for calculating the similarity between the new bug report and previously committed messages by developers. The features are collected from bug report commit messages and source code API documentation. Finally, total twenty two (22) features were extracted to identify the appropriate feature combination for training the bug report assignment model. Naive aggregation, Ordinal regression and Learning-to-rank are used to combine the extracted features. Sometimes only the experienced developers have permission to commit, but the resolution may be done by other developers, therefore detail of all commit messages need to be considered.

Jiang et al. (2020) developed a security bug report detection model known as LTRWES, by combining learning to rank with word embedding. LTRWES detects security related bug reports using four steps which are - Learning to Rank, Selecting, Training and Predicting. The Learning to Rank phase starts with ranking the NSBRs based on their similarity with the SBRs for correctly labeling all the bug reports. In this regard, the summary and description fields of each bug report is extracted and pre-processed. The pre-processing step includes text tokenization, lowercasing letters, stop word removal and stemming. Next, the similarity score between a pair of NSBR and SBR is calculated using BM25Fext technique. The similarity score is represented using a matrix where the rows indicate NSBRs and columns indicate SBRs. The average of the similarity scores in a row represents the actual similarity score of a NSBR with respect to the other SBRs. The NSBRs are then ranked based on their average similarity scores where the ranking shows the lower similarity scored bug reports at the top. The selection phase identifies the appropriate NSBRs by applying either of Multiple Selector (ms-selector) or Roulette Wheel Selection (rs-selector) algorithm on the previously ranked NSBRs. The ms-selector algorithm selects the top K lower scored bug reports from the ranked NSBRs. On the other hand, re-selector algorithm also selects K bug reports based on the probability of dissimilarity between the NSBRs and SBRs. The value of K is specified by multiplying the ratio value with the number of SBRs. Next, the top K selected NSBRs and all the SBRs are feed to the prediction model for training. The training phase first creates a vector representation of each bug report using the Continuous Bag of Words (CBOW) model. Naive Bayes, Multilayer Perceptron, Random Forest, K- Nearest Neighbor (KNN), Logistic Regression, and Support Vector Machine (SVM) techniques are used to train a model using the bug report vectors. Finally, in the prediction phase, when a new bug report comes, the bug report is also pre-processed and converted into a vector representation using the CBOW model. The model takes the vector representation as input and predicts the bug report as NSBR or SBR. The model is tested both on within project and cross project. But while testing cross projects, only one project was taken as cross project. For multiple cross projects how the technique performs can be analyzed further.

Akilan et al. (2020) proposed a computational efficient double tier duplicate bug detection system. The overall technique is divided into two phases – clustering and classification. For each bug report, structured (component, product, priority) and unstructured (summary, description) information's are extracted. The clustering phase prepossesses the bug reports and removes the redundant information. Next, the bug reports are clustered using Latent Dirchilect Allocation (LDA) topic modeling. When a new bug report comes, the Top N clusters which have similarity with the bug report are extracted. The selection of Top N clusters reduces the necessity of matching a bug report to all existing reports. As a result computational efficiency is achieved. In the classification phase, the bug reports are represented in vector formats using Word2Vec, GloVe and FastText. The similarity between these vectors are calculated using cosine similarity and Euclidean similarity. The proposed technique achieved 67% recall rate with 3 times less computation. However, the recall rate of topic modeling based techniques seems lower than exiting machine and deep learning techniques.

Kumar et al. (2020) developed a classification technique for identifying duplicate bugs. The technique first extracts the categorical (product, component and version) and textual (headline and description) features and preprocesses those. In feature generation phase, three types of features have been calculated such as text statistical, semantic and contextual feature. These features are fed to train the machine learning classification which uses the XGBoost algorithm. When new bug report arrives three servers known as App server, Model server and Embedding server works together to get the new report as input and classify the report as duplicate. The developed model was tested on Mozilla and Cisco project which achieved 90%, 98%, 94% and 87% precision, recall, F1-score and accuracy respectively. Although the paper used textual, categorical and customized extracted features, it did not mention which feature have highest impact in identifying appropriate duplicates.

Kukkar et al. (2020) proposed an automatic approach for detecting and classifying duplicate reports based on deep learning. The proposed approach considered the textual features such as title, summery and description and identical features such as bug id. Deep learning-based model mainly CNN is applied for eliciting the word's semantic and morphological relationship for the textual similarity assessment between bug reports. The proposed model achieved higher accuracy rate in between 85% - 99% and recall rate is 79%-94%. The proposed technique did not consider the contextual and temporal features which could impact the performance of CNN feature extraction.

By combining the attention based and context based feature, a duplicate bug report detection has been proposed by Rocha and Carvalho (2021). The overall technique is divided into three phases – training, retrieval and classification. In training phase, the bug report textual keywords are weighted for topic extraction and one hot encoding is created for contextual features. Next, a quotient loss function is devised for calculating the similarity between bug report embedding. The technique has been applied on Eclipse, NetBeans and Mozilla database which achieved accuracy of 84% accuracy. The application of the proposed loss function on closed or industrial project can be another future research scope.

Mahfoodh and Hammad (2022) proposed a duplicate bug report detection technique for predicting the risk factor of software components. The techniques uses the title / summary and description of bug report and extracts the word tokens. Next, the word tokens are converted into array which are fed into neural network. The similarity is measured using two approaches. The first similarity approach iterate on the words of one bug report to find its similar word with another bug report using Euclidean distance. On the other hand, the second similarity measure iterate on the words however within a given range. The technique achieved an average precision of 99.89%. The proposed techniques considers a fixed given range value for checking words. The increasing number of range value may affect the computational performance of the technique.

Few existing papers have also presented survey of the above mentioned duplicate bug report detection techniques (Neysiani et al., 2019a). Most of the papers only analyzed the different algorithms and evaluation metrics used for developing the duplicate detection techniques. However, none of the existing papers analyzed the effects of datasets, pre-processing techniques and bug report features while developing the detection techniques.

The difference in considering different datasets, pre-processing techniques, bug report features, algorithms and evaluation metrics raises the issue of generality of existing duplicate report detection approaches. Therefore, the detailed analysis regarding selection of input datasets, pre-processing mechanisms, features, algorithms and evaluation metrics may uncover the areas of improvement for future research.

4. Input Data Set

Open source systems are developed with contribution of developers from various location of the earth. Besides, now-a-days most of the software systems collect feedback from users to provide continuous support and maintenance. As a result, software bugs are reported by developers, testers and also users all over the world. To track and maintain these huge load of information, the open source software systems use bug tracking repositories such as – Jira, Bugzilla etc. Existing papers have used bug data set from the open source bug tracking systems (Ebrahimi et al., 2019; He et al., 2020; Neysiani et al., 2020; Tian et al., 2016). Some techniques have also incorporated software source codes along with the bug reports for better identification of duplicate bugs.

RQ1: What type of data are being used for duplicate bug report detection?

The most common data used for bug assignment, localization, classification and duplicate identification is bug repository. A number of open source bug repositories are available now-a-days. The bug repositories provide a number of functionalities to easily search and view the bug reports. Besides, the bug tracking systems have also provided different end-user and server-side utilities as third party extensions for easy and public access to submitted bug reports. Therefore, almost all of the existing techniques have used the open source bug tracking systems to collect input data. Many open source software projects uses these bug tracking systems for maintaining their bug repositories.

Figure 2 have listed the open source projects that have been used by existing duplicate bug detection techniques. The figure depicts that open source systems which maintain the bug repository consistently are used by most of the techniques. For example Eclipse is used by most of the existing techniques as a data set. Eclipse bug repository is maintained using Bugzilla. Bug reports can be searched and extracted in different formats. For example - JDT, SWT, ANT, UI are some of the products of Eclipse. Tian et al. (2016) and Alkhazi et al. (2022) have applied their proposed techniques on Eclipse JDT, SWT and UI product specific bug reports for evaluation.

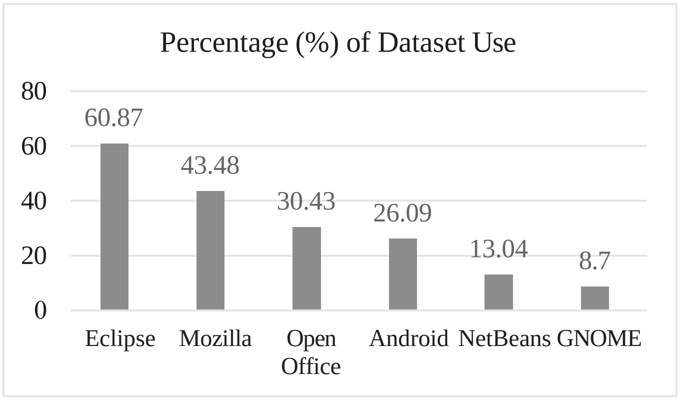


Figure 2: The usage of bug dataset by existing techniques

Besides Eclipse dataset, the Mozilla, Open Office, Android, NetBeans and GNOME are one of the most used dataset by most of the existing techniques (Ebrahimi et al., 2019), (He et al., 2020), (Sabor et al., 2017). Bugzilla has provided different plugins and websites for managing bug data of projects such as Bugzilla GNOME, Bugzilla Mozilla etc. (Bugzilla, 2021). Yuan et al. has collected security specific bug reports from Chromium, Derby, Camel, Wicket and Ambari dataset. In order to download the reports an open source web crawler known as Scarpy has been used (Zou et al., 2018). The tool is implemented in python for specifically crawling security specific bug reports using multi-type feature analysis.

For learning a model substantial amount of fixed bug reports are required. Yuan et al. collected minimum of 1000 bug reports from Ambari project for developing a security bug detection model (Jiang et al., 2020). The reason of selecting a lower number of bug reports is the availability of security bug reports. On the other hands, most of the techniques have used more than 10,000 bug reports as data set. Xiao et al. collected 2,73,710 bug reports from Eclipse bug repository for developing a heterogeneous information network to detect duplicate bugs (Xiao et al., 2020). The bug tracking systems enable the easy submission and maintenance of large scale bug reports. Therefore, it is evident that most of the techniques have been developed and tested on data set which are available in open source bug tracking systems and which already have large scale bug data set.

For ensuring the effectiveness, a duplicate bug report detection technique needs to be tested on both open source and closed source projects. The process of bug detection, bug reporting, developer involvement for bug resolution etc. differs between open source and closed source projects. Only a few techniques have applied closed source projects for development and evaluation (Cooper et al., 2021a, 2021b).

Bug reports in open source systems are reported using bug tracking systems. Each bug tracking system provides a predefined form structure for reporting bug reports. Bugs are reported using these forms and can be downloaded using plugins provided by the bug tracking systems (Bugzilla, 2021). By using the plugins the bug reports can be downloaded in XML, CSV and plain text format. Figure 3 shows the sample Eclipse bug report having id 519169 in XML format.

```
<bug>
        <id>519169</id>
        <creation_ts>2017-07-04 10:54:44 -0400</creation_ts>
        <summary>Vulnerability found in Eclipse</summary>
        <product>andmore</product>
        <component>Core</component>
        <version>0.5.0</version>
        <status>ASSIGNED</status>
        <priority>P1</priority>
        <severity>critical</severity>
        <long_desc><commentid>2849292</commentid>
                <who name="Alon Boxiner">alonbo</who>
             <thetext>Steps to reproduce: 1. Open a new Android
                                            project.</thetext>
        </long_desc>

</bug>
```

Figure 3: Sample eclipse bug report in XML format

Few approaches have used video-based bug reports as input (Cooper et al., 2021a, 2021b). These approaches leveraged screen-recording features of Android and iOS device for capturing the video of bugs. The videos showing error are reported as bugs for further resolution. The videos are converted into a consecutive series of images. Next, the text of the images are extracted and matched with incoming bug reports for duplicate detection.

The source code repository are collected from version control systems such as GitHub, Bit bucket etc. Each version control system provides command line functionalities to download the repository. Besides, the associated commit messages can also be extracted using the terminal commands. The comments of commit messages can be extracted in plain text, XML etc. formats. The change history of source code files can also be extracted from commit details.

The above discussion answers the RQ1 by identifying that most of the existing techniques generally use fixed bug reports as input data set. Besides, open source systems which have maintained rich bug report repository (such as Eclipse, Mozilla, NetBeans etc.) for long time are also preferred by most of the techniques. Along with the bug reports, the source code, commit files, version history and comments are also used as input data. Now-a-days the video and screenshot based bug reports are also being used. Thus the study of video or screenshot based duplicate bug detection can be explored by researchers. Although most of the technique has used open source bug repository, only a few approaches have tested the technique on closed source projects. Therefore, future research scope lies in finding appropriate duplicate detection technique for closed source and cross projects.

5. Feature Selection and Pre-processing

After collecting the input bug reports, appropriate bug report property or feature needs to be selected for learning the duplicate detection technique. The more appropriate the property is selected, the more relevant duplicate can be identified.

RQ2: Which bug report properties are used for duplicate bug report detection?

The bug report properties can be also referred as bug report features where each feature indicates a new aspect of the reported bug. Based on the content type and previous usage, the features are divided into four major categories which are - Textual, Categorical, Contextual and Temporal.

5.1 Textual Feature

The textual feature refers to the bug report properties which are written in natural language format. The title, summary, description and comments are the main textual properties of a bug report as shown in Table 1. While reporting a bug, the developers generally add a short title/summary of the bug report which is written in natural language format. Besides the title, a detailed description of the bug is also added. The description often contains source code stack traces. As a result, the description of bug report may be of any length. After the reporting of the bug, developers interact with each other by posting comments during the bug resolution. Hence, the comment property also contains text in natural language format.

As the textual fields are written in natural language, so these fields indicate appropriate developer’s or user’s perspective about a reported bug. Table 1 shows that 20 out of the 23 papers have used description field of the bug report as feature for duplicate bug detection. Next, the title or summary is the most used bug report feature. The table also shows that out of the four major categories, textual features are the most used bug report features.

Table 1: Features of Bug Report

Feature Type	Feature Name	No of Times Used
Textual	Description	20
	Title / Summary	18
	Comment	3
Categorical	Component	12
	Product	11
	Priority	10
	Operating System	7
	Version	5
	Severity	4
	Hardware	1
	Status	1
Contextual	Topic of report extracted based on textual features	7
Temporal	Open / Close Date	4
Identical	Bug Id	5
Structural / Attachment	Source code / file attachment	4

5.2 Categorical Feature

The categorical features represent the bug report properties which value is selected from a list of predefined values. It includes the component, product, priority, operating system, version, severity and status of the bug report. The similarity between two duplicate reports are checked by calculating the equality of the features. Table 1 shows that categorical features are the second highest used features by the existing techniques. The Product and Component are the most used features of categorical type.

5.3 Contextual Feature

Unlike the textual and categorical features, the contextual features tend to identify inherent topic of the bug report using topic modeling techniques (Rocha & Carvalho, 2021). These features are calculated by measuring the similarity between the content of the bug report and a predefined list of words corresponding to specific topics (such as security, performance, enhancement etc). The contextual similarity of bug reports are measured using different techniques such as Cosine similarity, Manhattan distance, LDA etc.

5.4 Temporal Feature

The temporal features tend to check the recency between the bug reports in terms of reporting or closing time. These features are calculated by taking the subtraction value of same fields from two different bug reports. The features are less used in comparison of the other features as shown in Table 1. However this feature can be helpful in filtering the recent bug reports. As a result, the search space of duplicate report checking can be reduced.

Apart from the above mentioned features, Identical, Structural and Derived features are also used in existing techniques (Mahfoodh & Hammad, 2022). The identical features check the distance between the unique id of bug reports to understand their reporting sequence. Source code patch or files are sometimes attached with the bug report (Wang et al., 2008). These file attachments are considered as structural features while duplicate detection. Derived features are calculated by applying the TF-IDF, BM25F and date interval calculation techniques on categorical and textual features of the bug report.

After extraction of features, ranking or combining features has also been done by few proposed techniques (Alkhazi et al., 2020, Neysiani et al., 2020). For ranking the features Naive aggregation, Ordinal regression and Learning-to-rank have been used in literature. Few existing works have assigned specific value to features for ranking. The values are assigned based on weighted average of information gain ratio, Gini index, chi-square, PCA of the features. The more appropriate feature is selected, the more appropriate duplicate reports can be identified. Therefore, techniques for ranking and identifying effective features can be explored further.

Since the bug report fields are stored in different formats as discussed above, therefore before checking similarity the data needs to be normalized in general format. Figure 4 shows the popular pre-processing techniques which have been used by researchers. It can be seen that text tokenization and stop word removal have been used 78.26% times in the 23 studied papers. Stemming is the third most used preprocessing technique being used in 43.48% cases. Jalbert et al. have applied MontyLingua tool, ReqSmile tool and Porter Stemming algorithm for tokenization, stop word removal and stemming respectively (Jalbert & Weimer, 2008). Lower case conversion, n gram word conversion, lemmatization are also used while pre-processing the text fields of the bug report. For stack trace or source code files, programming specific word removal, file path replacement mechanism are applied (Kumar et al., 2020).

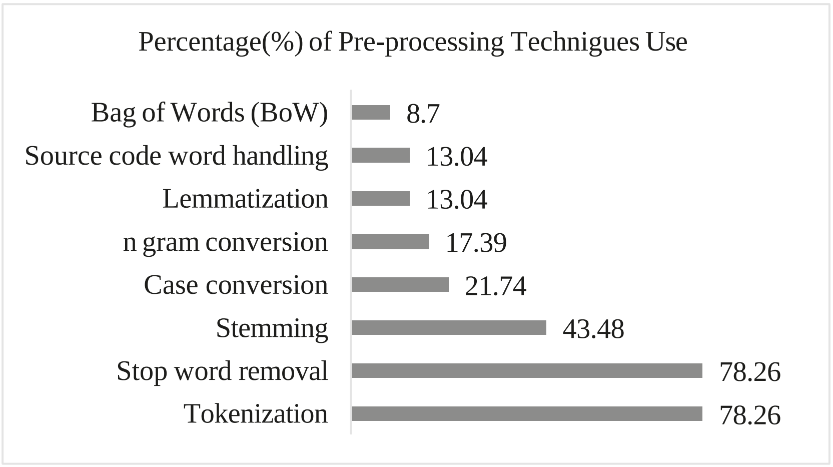


Figure 4: Sample eclipse bug report in XML format

Based on the above discussion, the answer to research question RQ2 depicts that textual and categorical properties are the most used properties of a bug report. On the other hand, stop word removal, tokenization and stemming are the popular text cleansing techniques. However, there is no clear discussion on among the four features, which are the most dominant one in identifying duplicates. Therefore, the effect of using individual and combined features in identifying duplicate bug reports can be studied in future.

6. Learning Algorithms

Once appropriate features are selected and pre-processed, the features are fed into different algorithms to develop a model that can detect duplicate bug reports.

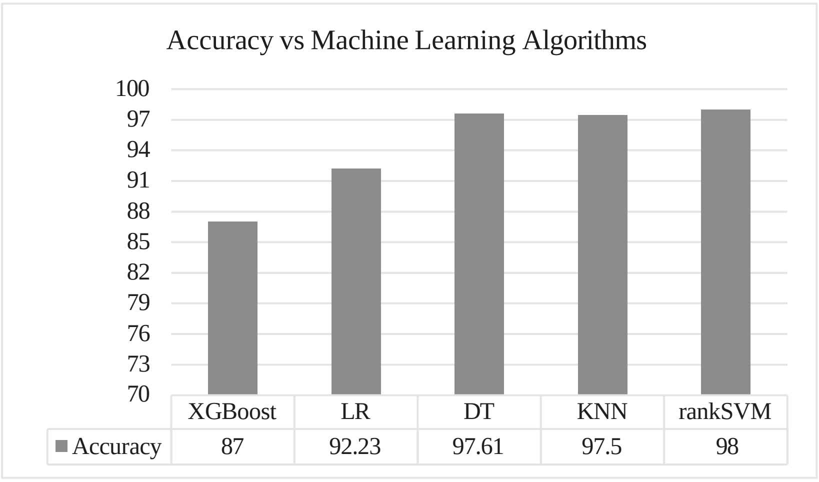
RQ3: What type of algorithms are being used for duplicate bug report detection?

The existing techniques have used Machine Learning (ML) (Neysiani & Babamir, 2019c), Information Retrival (IR) (Sabor et al., 2017) and Deep Learning (DL) (Poddar et al., 2019) based techniques for identifying duplicate reports. Information retrieval based techniques generate a ranked list of duplicates corresponding to an incoming report (Sabor et al., 2017; Johannes & Mira, 2013) .On the other hand, machine learning and deep learning based techniques classify a incoming bug report as a duplicate (Akilan et al., 2020; Kukkar et al., 2020; He et al., 2020). Some of the existing work have also combined these techniques in different phases of the duplicate detection algorithm (Neysiani & Babamir, 2020).

For evaluation of the existing techniques, researchers have used different metrics such as Accuracy, Recall, Precision, F1- score etc. Among these, accuracy and recall has been used by most of the techniques (Neysiani et al., 2020; He et al., 2020; Xiao et al., 2020; Neysiani & Babamir, 2020, 2019b). Accuracy refers how many correct classification have been predicted by the model as shown in Equation 1. On the other hand, recall refers how many correct duplicates have been placed in the ranking from the actual duplicates as shown in Equation 2.

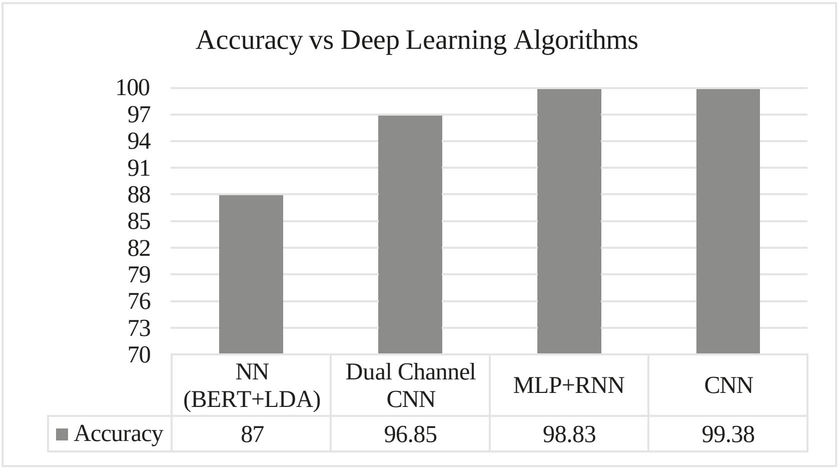
$$Accuracy = \frac{True\ Prediction}{Total} \tag{1}$$

$$Recall = \frac{True\ Duplicates}{Actual\ Duplicates} \tag{2}$$



	XGBoost	LR	DT	KNN	rankSVM
■ Accuracy	87	92.23	97.61	97.5	98

Figure 5: Analysis of accuracy in different Machine Learning (ML) algorithms



	NN (BERT+LDA)	Dual Channel CNN	MLP+RNN	CNN
■ Accuracy	87	96.85	98.83	99.38

Figure 5 and 6 shows the accuracy of duplicate bug detection technique using different Machine Leaning and Deep Leaning algorithms respectively. The reason behind selecting these algorithms is the 23 reviewed papers have used these algorithms in combination or individual. Among the different ML algorithms the accuracy of Decision Tree (DT), KNN and rankSVM is above 97% which is prominent. On the other hand, the deep learning algorithms such as Recurrent Neural Network (RNN) and Convolutional Neural Network have accuracy above 98% which shows DL algorithms have better accuracy than ML algorithms. However, both of the algorithms have reached above 96% accuracy while detecting duplicate bug reports which indicate that there is small scope of future improvement in terms of accuracy. However, while achieving the accuracy the performance in terms of time, computation can be considered as future research scope. Besides the results of Figure 5 and 6 have been achieved using open source bug repositories where huge volume of data is available. The accuracy of these algorithms in terms of closed source bug repositories need further attention.

Table 2: Analysis of Recall in Information Retrieval, Machine Learning and Deep Learning Techniques

Information Retrieval Techniques			
LDA	TF-IDF	BM25F	LCS
67	84	93.04	97.27
Machine Learning Techniques			
LR	KNN	XGBoost	DT
92.95	97.51	98	99.94
Deep Learning Techniques			
NN	CNN	Dual Channel CNN	RNN
80	91.48	96.7	97.07

Table 2 shows the recall value of different Information Retrieval, Machine Learning and Deep Learning algorithms. Among the three category, IR based algorithms have lowest recall of 67% and 84% using Latent Dirichlet Allocation (LDA) and TF-IDF technique respectively. Jalber et al. also mentioned in the paper that TF-IDF achieved recall rate of 52% which is poor (Jalbert & Weimer, 2008). On the other hand, Machine Learning and Deep Learning techniques have higher recall in terms of IR based techniques. Decision Tree has accuracy of 99.94% in detecting duplicate bug reports (Soleimani & Morteza, 2020). The high value of recall for ML and DL techniques represent the limited scope of improvement in this metric. The future scope lies in evaluating the performance of these algorithms in terms of time, computation, memory usage etc. to reach this recall value.

The above discussion answers the research question RQ3 by identifying the fact that ML and DL algorithms achieve higher accuracy and recall than the IR based techniques. The higher value of these metrics show little scope of improvement in these metrics. However, in future the performance of combined application of these algorithm can be analyzed. Another future scope for research can be the implementation of duplicate detection technique as plugin for the software development IDEs. As a result, before reporting a bug, the developers can check for its duplicate bugs.

7. Conclusion

With the increasing of duplicate bug report submission, the need for appropriate duplicate bug report detection has become important. A general duplicate detection technique consists of three steps - *Feature Selection, Bug Report Pre-processing* and *Learning Algorithm or Model*. Based on this, a number of duplicate detection techniques have already been proposed by researchers. Therefore, this paper discusses the present literature work of duplicate bug report detection. In order to do so the papers devises three research questions which tend to analyze the input data set, the feature selection and prepossessing, and the evaluation of different learning algorithms respectively. With each research question, the future road map for research in duplicate detection has also been enlisted.

References

Akilan, T., Shah, D., Patel, N., & Mehta, R. (2020). Fast detection of duplicate bug reports using lda-based topic modeling and classification. In proceedings of the ieee international conference on systems, man, and cybernetics (smc) (pp. 1622–1629).

Alkhazi, B., DiStasi, A., Aljedaani, W., Alrubaye, H., Ye, X., & Mkaouer, M. W. (2020). Learning to rank developers for bug report assignment. Applied Soft Computing, 95 (pp. 106667).

Anvik, J., Hiew, L., & Murphy, G. C. (2005). Coping with an open bug repository. In proceedings of the oopsla workshop on eclipse technology exchange (pp. 35–39).

Bugzilla. (2021). Bugzilla addons. Retrieved from https://wiki.mozilla.org/Bugzilla:Addons (Last accessed 23 October 2021)

Cooper, N., Bernal-Cardenas, C., Chaparro, O., Moran, K., ´ & Poshyvanyk, D. (2021a). It takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In proceedings of the ieee/acm 43rd international conference on software engineering (icse) (pp. 957–969).

Cooper, N., Bernal-Cardenas, C., Chaparro, O., Moran, K., & ´ Poshyvanyk, D. (2021b). A replication package for it takes two to tango: Combining visual and textual information for detecting duplicate video-based bug reports. In proceedings of the ieee/acm 43rd international conference on software engineering: Companion proceedings (icse-companion) (pp. 160–161).

Ebrahimi, N., Trabelsi, A., Islam, M. S., Hamou-Lhadj, A., & Khanmohammadi, K. (2019). An hmm-based approach for automatic detection and classification of duplicate bug reports. Information and Software Technology, 113, (pp. 98–109).

Education, C.R.(2021).Core website. Retrieved from http://portal.core.edu.au/conf-ranks/ (Last accessed 19 November 2021)

He, J., Xu, L., Yan, M., Xia, X., & Lei, Y. (2020). Duplicate bug report detection using dual-channel convolutional neural networks. In proceedings of the 28th international conference on program comprehension (icpc) (pp. 117–127).

Jalbert, N., & Weimer, W. (2008). Automated duplicate detection for bug tracking systems. In proceedings of the ieee international conference on dependable systems and networks with ftcs and dcc (dsn) (pp. 52–61).

Jiang, Y., Lu, P., Su, X., & Wang, T. (2020). Ltrwes: A new framework for security bug report detection. Information and Software Technology, 124, (pp. 106314). Johannes, L., & Mira, M. (2013). Finding duplicates of your yet unwritten bug report. In proceedings of the 17th european conference on software maintenance and reengineering (pp. 69–78).

Kukkar, A., Mohana, R., Kumar, Y., Nayyar, A., Bilal, M., & Kwak, K.-S. (2020). Duplicate bug report detection and classification system based on deep learning technique. IEEE Access, 8, (pp. 200749–200763).

Kumar, A., Madanu, M., Prakash, H., Jonnavithula, L., & Aravilli, S. R. (2020). Advaita: Bug duplicity detection system. arXiv preprint arXiv:2001. (pp. 10376).

Lee, C.-P., & Lin, C.-J. (2014). Large-scale linear ranksvm. Neural computation, 26(4), (pp. 781–817).

Mahfoodh, H., & Hammad, M. (2022). Identifying duplicate bug records using word2vec prediction with software risk analysis. International Journal of Computing and Digital Systems, 11(1), (pp. 763–773).

Neysiani, B. S., & Babamir, S. M. (2019a). Duplicate detection models for bug reports of software triage systems: A survey. Current Trends In Computer Sciences & Applications, 1(5), (pp. 128–134).

Neysiani, B. S., & Babamir, S. M. (2019b). Improving performance of automatic duplicate bug reports detection using longest common sequence: Introducing new textual features for textual similarity detection. In proceedings of the 5th international conference on knowledge based engineering and innovation (kbei) (pp. 378–383).

Neysiani, B. S., & Babamir, S. M. (2019c). New methodology for contextual features usage in duplicate bug reports detection: dimension expansion based on manhattan distance similarity of topics. In proceedings of the 5th international conference on web research (icwr) (pp. 178–183).

Neysiani, B. S., & Babamir, S. M. (2020). Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches. In proceedings of the 6th international conference on web research (icwr) (pp. 288–293).

Neysiani, B. S., Babamir, S. M., & Aritsugi, M. (2020). Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. Information and Software Technology, 126, (pp. 106344).

Poddar, L., Neves, L., Brendel, W., Marujo, L., Tulyakov, S., & Karuturi, P. (2019). Train one get one free: Partially supervised neural network for bug report duplicate detection and clustering. arXiv preprint arXiv:1903. (pp. 12431).

Rahman, M. M., Khomh, F., & Castelluccio, M. (2020). Why are some bugs non-reproducible?:–an empirical investigation using data fusion–. In proceedings of the 36th international conference on software maintenance and evolution (icsme) (pp. 605–616).

Rocha, T. M., & Carvalho, A. L. D. C. (2021). Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information. IEEE Access, 9, (pp. 44610–44630).

Sabor, K. K., Hamou-Lhadj, A., & Larsson, A. (2017). Durfex: a feature extraction technique for efficient detection of duplicate bug reports. In proceedings of the ieee international conference on software quality, reliability and security (qrs) (pp. 240–250).

Serrano, N., & Ciordia, I. (2005). Bugzilla, itracker, and other bug trackers. IEEE software, 22(2), (pp. 11–13).

Soleimani, N. B., & Morteza, B. S. (2020). Effect of typos correction on the validation performance of duplicate bug reports detection. In proceedings of the 10th international conference on information and knowledge technology (ikt), tehran, iran (pp. 1–2).

Tian, Y., Wijedasa, D., Lo, D., & Le Goues, C. (2016). Learning to rank for bug report assignee recommendation. In proceedings of the 24th international conference on program comprehension (icpc) (pp. 1–10).

Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In proceedings of the 30th international conference on software engineering (pp. 461–470).

Xiao, G., Du, X., Sui, Y., & Yue, T. (2020). Hindbr: Heterogeneous information network based duplicate bug report prediction. In proceedings of the ieee 31st international symposium on software reliability engineering (issre) (pp. 195-206).

Zou, D., Deng, Z., Li, Z., & Jin, H. (2018). Automatically identifying security bug reports via multitype features analysis. In proceedings of the Australasian conference on information security and privacy (pp. 619–633)