J Inf Process Syst, Vol.15, No.4, pp.967~985, August 2019
https://doi.org/10.3745/JIPS.04.0130

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Systematic Review of Bug Report Processing Techniques to Improve Software Management Performance

Dong-Gun Lee* and Yeong-Seok Seo*

Abstract

Bug report processing is a key element of bug fixing in modern software maintenance. Bug reports are not processed immediately after submission and involve several processes such as bug report deduplication and bug report triage before bug fixing is initiated; however, this method of bug fixing is very inefficient because all these processes are performed manually. Software engineers have persistently highlighted the need to automate these processes, and as a result, many automation techniques have been proposed for bug report processing; however, the accuracy of the existing methods is not satisfactory. Therefore, this study focuses on surveying to improve the accuracy of existing techniques for bug report processing. Reviews of each method proposed in this study consist of a description, used techniques, experiments, and comparison results. The results of this study indicate that research in the field of bug deduplication still lacks and therefore requires numerous studies that integrate clustering and natural language processing. This study further indicates that although all studies in the field of triage are based on machine learning, results of studies on deep learning are still insufficient.

Keywords

Bug Report, Clustering, Duplication Detection, Information Retrieval, Machine Learning, Priority, Severity, Software Developer Assignment, Software Management, Triage

1. Introduction

With the constantly increasing complexity of bug report processing because of modern developments, developers are unable to avoid bugs. A bug simply means any unexpected, result and they significantly reduce the quality of software if unfixed. Thus, bug fixing is an extremely important task for developers.

In a world of modern developments, developers primarily address bugs through bug reports. A bug report is a description of what a quality assurance team or users think is a discrepancy between the actual and the promised outcomes of a software. Addressing bug reports is indispensable in software maintenance, and thus, software engineers expect a bug tracking system to be efficient [1,2].

Bug report processing involves the processes from submitting the bug report [2-5], to fixing the bug. Fig. 1 shows how bugs are addressed at each stage of processing [6]. As seen from Fig. 1, many stages of processing are involved before the "Fixed" phase is reached. "Assigned" stage involves the most important task of allocation of bug reports to appropriate developers, and thus, requires additional effort than other stages [7-9]. There are two main reasons for this.

Manuscript received February 1, 2019; first revision April 9, 2019; accepted April 18, 2019.
Corresponding Author: Yeong-Seok Seo (ysseo@yu.ac.kr)
* Dept. of Computer Engineering, Yeungnam University, Gyeongsan, Korea (dklee777777@naver.com, ysseo@yu.ac.kr)

First, a duplicate bug report is a report on bugs that have already been solved. The amount of duplicate bug reports/whole bug reports is up to 30% in Firefox [10]. Thus, reassigning the already solved bug reports to developers is a waste of efficiency. Therefore, it is desirable to bundle these duplicate bug reports as one set. Generally, automatic duplicate bug report identification techniques do not work perfectly, and therefore, bug report identification is performed manually.

Second, a bug report triage is the classification of bug reports by their meta field, such as priority or severity. Numerous bug reports are submitted daily and it is impossible for developers to process all of them. Thus, developers need to first address the bugs that are fatal to systems or make users uncomfortable. This indicates that a bug report triage is essential for software maintenance. Because the duplicate bug report identification process is included in bug report triage, bug report triage is also performed manually. In the case of Eclipse, two man-hours are required every day [11]. To compensate for this, many triage automation techniques have been proposed, but they do not show satisfactory accuracy.

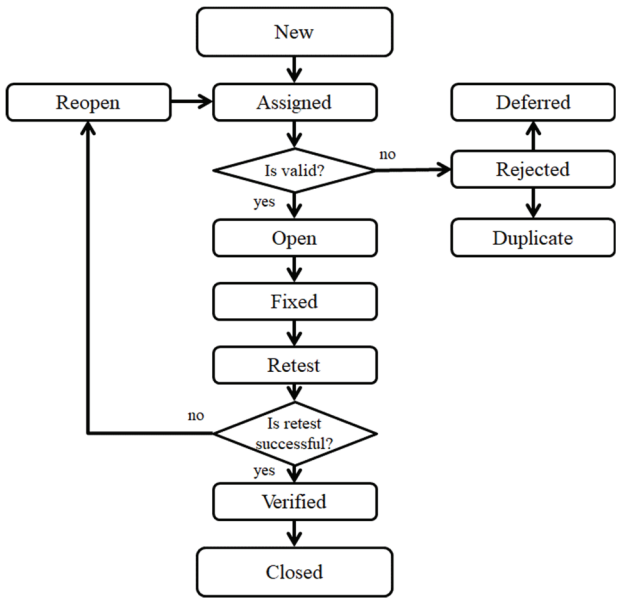

Fig. 1. Bug report processing cycle [3].

In this study, we investigate the bug report processing techniques based on three factors: utilized techniques, experiment target, and comparison with two fields (duplicate bug report identification and triage accuracy improvement) to improve the software maintenance performance. In the case of duplicate bug report identification, we focus on topic modeling, natural language processing, information retrieval, and clustering based on similarity. Most deduplication studies tend to utilize Mozilla (or a software developed by Mozilla, such as Firefox), Eclipse, and Open Office to configure their testing set. A few of them have proposed methods that integrate 2–3 techniques. In the case of improving bug report triage accuracy, we focus on classification techniques such as k-nearest neighbors (KNN), naïve Bayes (NB), and support vector machine (SVM).

The rest of this paper is organized as follows: Section 2 introduces the form of bug reports in a modern development environment. Section 3 provides studies identifying duplicate bug reports. Section 4

presents studies for improving bug report triage accuracy similar to their meta field such as priority or severity. Finally, Section 5 concludes the paper and discusses some possible future issues.

2. Bug Report Features in Modern Software Development Environment

2.1 Form of Bug Reports

A bug report is a document intended for users or a Q&A team to communicate bugs that cause program failures to developers in a specific format. Figs. 2–4 show reports of a bug in JIRA [12], Mantis [13], and Bugzilla [14]. The bug report of Fig. 2 is for Block Chain. It was submitted by the reporter using "Mekia Edwards" as the identifier. It can be seen that it is classified as having a "Highest" priority. It is an Unresolved status. The bug report of Fig. 3 is for the MantisBT project. It was submitted by the reporter using "shashi1" as an identifier. It can be seen that it is classified as having an "emergency" priority and a severity of "important". It is the Opened status. The bug report of Fig. 4 is for Android. It was submitted by the reporter using "philipp" as an identifier.

The metadata includes the following components:

1) Product: Describes the environment in which a bug occurs.
2) Component: Classifies the items based on their closeness to meta fields such as priority and severity. These metadata can be used as criteria for classifying bug reports.

Priority is the classification of a bug over others based on its importance, and severity indicates the critical nature of a bug; thus, priority and severity are the most important tasks in bug classification, which is simply called "Triage". The accuracy of triage is directly related to the quality of the software used for classification. In the case of Bugzilla, priority is in a scale of p1 to p5, with p1 being the highest priority. In the same case, severity is classified as Blocked (fatal bug rendering development or patching of a software impossible), Critical (fatal bug affecting program execution), Major (important functional flaw), Normal (common error), Minor (not an important functional flaw, or is an error can be easily fixed), Trivial (typographical error), and Enhancement (function and performance improvements).

2.2 Bug-Handling Process

Once a bug appears, it is processed according to the bug report processing cycle, as presented in Fig. 1. A bug is addressed at each stage as follows:

- **New**: A new bug is first logged or found. A new bug is usually found by a user or the QA team. It is then moved to the "Assigned" stage where it will be classified to determine the type of handling.
- **Assigned**: After a tester logs a bug, this stage executes the triage. Triaged bugs are classified by their priority and/or severity. If the triage determines a bug to be rejected, they are moved to the "Rejected" phase. If the triage determines a bug, it is assigned to the appropriate developer based on its priority and severity.
- **Open**: This stage analyzes and corrects the bugs assigned to developers. When the bug is fixed, it is moved to the "Fixed" stage.
- **Fixed**: In this stage, the developer sends the bug to the test team after fixing it. Based on the results of the test, the bug may be moved to the "Reopen" or "Retest" stages.

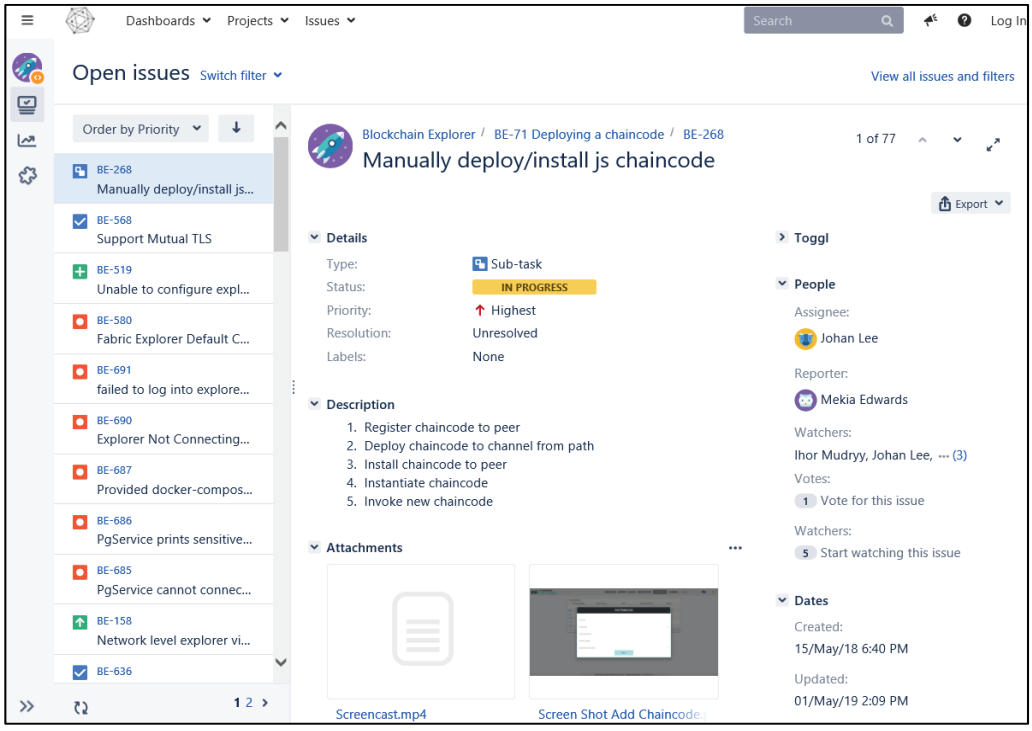

Fig. 2. Bug report in JIRA [12].

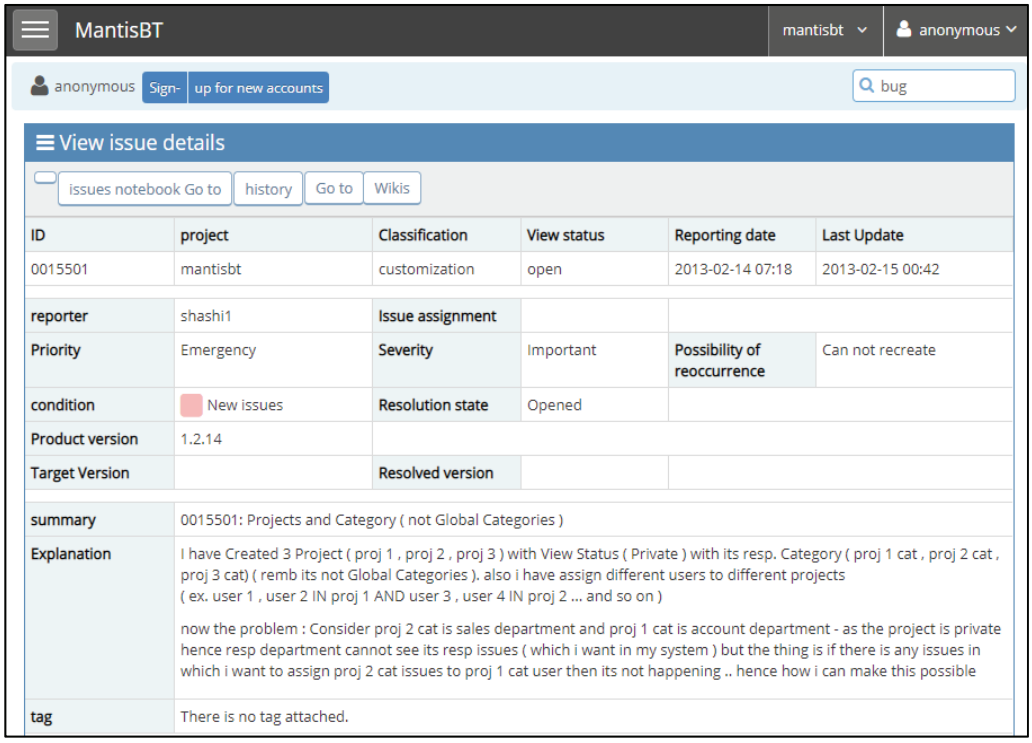

Fig. 3. Bug report in Mantis [13].

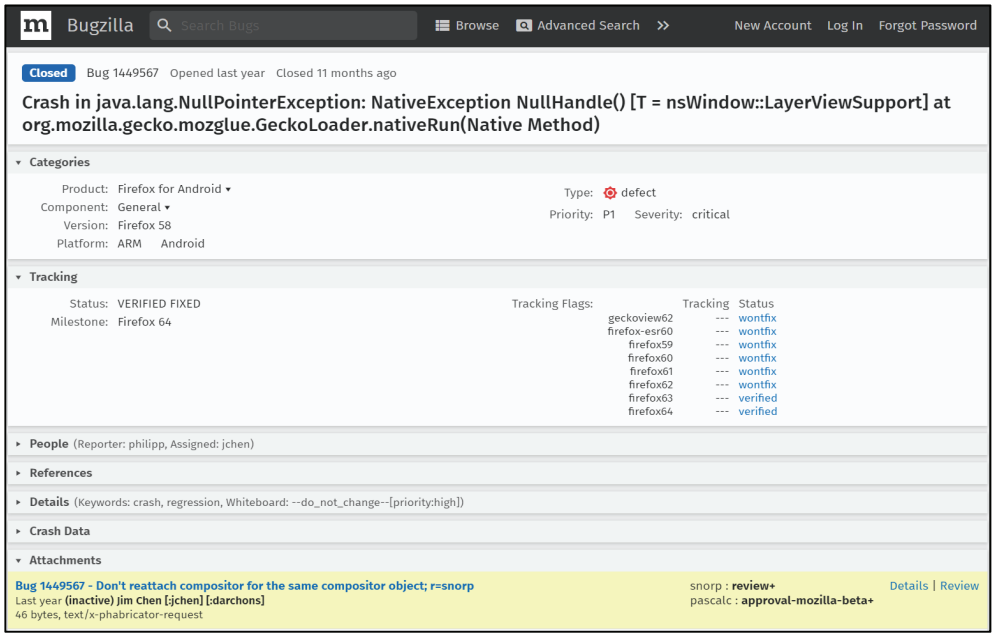

Fig. 4. Bug report in Bugzilla [14].

- **Reopen**: In this stage, if a bug fixed by a developer still exists, it is sent back to the "Assigned" stage and the succeeding stages follow.
- **Rejected**: If the developer feels the bug is not genuine, the bug is rejected. Then, the status of the bug is changed to "Rejected".
- **Deferred**: A bug assigned the "Deferred" status implies that it is expected to be fixed in next releases. There are several reasons for assigning the "Deferred" status to a bug. A few reasons are priority of the bug being low, lack of time for release, and no major effect by the bug on the software.
- **Duplicate**: If a bug is reported twice or if two bugs are found to cause the same problem, then the status of one of them is changed to "Duplicate".
- **Retest**: In this stage, the tester retests the modified code provided by the developer to check if the defect has been fixed.
- **Verified**: The tester retests the bug after it is fixed by the developer. If this bug is no longer detected in the software, then the status of the bug changed to "Verified".
- **Closed**: If the tester feels that the bug no longer exists in the software, the status of the bug is changed to "Closed". This state means that the bug has been fixed, tested, and approved.

2.3 Issues in Processing Previous Bug Reports

Typical bug report processing systems generally focus on the identification of the types of bug reports. Through this identification, they attempt to automatically classify the types of bug reports and assign software developers responsible for resolving the reported issues. Thus, the most important point is to be able to accurately identify the type of the bug report. However, there are several issues for accurate type identification. For example, a bug report could be classified as "Trivial" because fonts are not just rendered in the website. Conversely, a bug report could be classified as "Critical" because a system crash

occurred. Even though this classification scheme is not entirely incorrect, it could cause wrong classification of the bug reports. Suppose there is a font which is not rendered correctly. If there is a typo in this font, or the font is represented in the pop-up pages that would disappear in a couple of days, these bugs could be trivial. In contrast, the fonts in the graphical user interface (GUI), which are always displayed on the web page or in fonts that contain important content, can lead to a misunderstanding by the user. In these cases, these bugs could be critical types in spite of the problem being just the font. In practice, because software developers need advanced techniques to improve bug report processing efficiency, many studies are focusing on reducing duplicate bug reports and improving bug triage performance, with the correct identification techniques for types of bug reports.

3. Techniques for Reducing Duplicate Bug Reports

When a bug occurs, many users report the bug to developers. Eventually, developers receive a lot of bug reports on the same bug. Duplicate bug reports cause boredom to developers and decrease their work efficiency. Thus, duplicate bug reports can be grouped as one set and processed at a time. There are many techniques for reducing the number of duplicate bug reports. In this study, we introduce novel methods for deduplication in three important areas: natural language processing (NLP), information retrieval (IR), and similarity-based classification.

3.1 Natural Language Processing

A common way to identify duplicate bug reports is to use NLP [15,16]. Because bug reports are described in natural language used by humans, different types of words are generally used. Therefore, for greater accuracy, methods (such as stemming, lemmatization, and stopword processing) are used [17,18]. Among all the NLP techniques, topic modeling is mainly used, which processes words using summary or text from bug reports and selects the criterion words. The topic selected in this process represents a few features of the bug report and defines the bug report that has same topic as a duplicate bug report [19,20].

The most frequently used technique for topic modeling is latent Dirichlet allocation (LDA). LDA addresses the drawbacks of the probability model probabilistic latent semantic analysis (PLSA). LDA is similar to PLSA in that it obtains the probability of words in a document, but the distribution of topics is assumed to follow the Dirichlet distribution [21].

Baek et al. [22] identified duplicate bug reports using the LDA, NB [23], and NB polynomial [23]. They compared the obtained results with the conventional machine learning method using Eclipse bug reports. As a result, duplicate bug reports were identified with an accuracy of approximately 80%. In addition, they indicated significant differences in using statistical methods.

Zou et al. [24] proposed the LDA and N-gram (LNG) technique, which consists of two parts. One is topic modeling with existing LDA method and the other is a linearly coupled weight-based N-gram similarity. A new measure, exact-accuracy (EA) rate, was introduced to verify redundancy. As a result of verification using approximately 230,000 Eclipse bug reports, the LNG technique was found to have improved recall rate, precision rate, and EA rate compared with the existing techniques in detecting bug report duplication. In particular, the recall rate was improved by 2.96% to 10.53% compared with DTBM [16], which is a state-of-the-art approach used in detecting bug report duplication.

3.2 Information Retrieval

Identification of duplicate bug reports based on information retrieval uses the meta data in bug reports. The bug report contains various meta data such as bug type, operation system, priority, and severity, which help identify similar bug reports [25-27]. However, because even bug reports in the same environment can sometimes have different solutions, duplicate bug report identification techniques based on information retrieval are often used in conjunction with other methods.

Alipour et al. [15] focused on contextual information for bug report deduplication. They extracted the context implied by bug reports using BM25F techniques, a score algorithm used in IR fields, in pre-built software dictionaries (word lists). Their study also found that the extracted context from the bug report has non-functional requirements—related to software's quality closely (not functionality)—as well as the subject of the bug report. Alipour et al. [15] constructed the words list using Android layered architectural words [28], software non-functional requirements words [29], Android topic words using LDA [30], Android topic words using labeled-LDA [30], and random words from English dictionary and evaluated using 37,236 Android reports. The result was a more than 11.55% improvement in bug report redundancy compared with the result of Sun et al. [19].

Aggarwal et al. [31] supplemented the study of Alipour et al. [15] using a method called software literature context. This method uses words extracted from software engineering literature. This study provided a beneficial result using which the manual efforts on deduplication can be reduced, compared with the study by Alipour et al. [15] that extracts words from the bug report, and whose result was a slight loss of accuracy. The authors [31] further validated documents from Eclipse, Mozilla, Open Office, and Android bug reports used by Alipour et al. [15]. Their method was also shown to be available for general cases compared with Alipour et al.'s method.

Sun et al. [19] proposed REP, a technique using similarity between two different bug reports dataset. REP is an extension of BM25F and uses non-textual fields (component, product, and version) as well as text content such as summary and contents of bug reports. Using these measures, the bug reports were identified and evaluated using Mozilla, Eclipse, and Open Office's bug reports, resulting in a 10%–27% increase in recall rate compared with the previous version, and a 17%–23% improvement in mean average precision.

Sun et al. [17] proposed a discriminative model for searching similar bug reports in bug tracking systems. The model uses information retrieval techniques to determine similarity between two bug reports based on 54 features. They applied the model to three large software bug reports from Firefox, Eclipse, and Open Office, demonstrating a relative improvement of 17%–31%, 22%–26%, and 35%–43% over the-state-of-art techniques.

3.3 Clustering

The method of identifying duplicate bug reports based on similarity is highly reliable because each bug report is directly compared. This technique can also be used to classify bug reports using a scale called similarity and is often used together with clustering because of their high affinity [32].

Hiew [32] introduced an approach based on topical detection and tracking techniques that considered clustering in news articles. They used this approach for each Firefox, Eclipse, Fedora, and Apache projects to achieve a 29% accuracy and a 50% recall rate as the best result for Firefox.

Gopalan and Krishna [33] proposed a clustering-based technique to identify redundancy in a set of large bug reports. This technique maintains a low false positive, i.e., the rate at which normal bug reports are considered redundant. Eclipse, Mozilla, and Open Office were used for evaluation of this technique.

3.4 Hybrid Techniques

Achieve satisfactory identification accuracy with one technique is difficult. Therefore, many researchers have begun to use more than one technique in combination. In particular, information retrieval techniques are preferred to be supplemented mainly by the LDA method, which is a topical modeling technique of machine learning.

Nguyen et al. [16] proposed DBTM, a technique that has both the merits of topical-based features and IR-based features. DBTM used BM25F and T-Model, an extension of the topic modeling technique LDA. Evaluation of the DTBM technique using bug reports from Eclipse, Open Office, and Mozilla indicated increased duplicate bug report identification by approximately 20% compared with the Relational Topic Model (RTM) [34] and REP [19].

Lin et al. [35] proposed SVM-SBCTC method, an SVM that considers the semantic correlation of text based on SVM discriminative scheme (SVM-54) [17]. SVM-SBCTC applied all the correlation between the NLP (Word2vec), information retrieval (BM25), and clustering-based features. They verified the study on three large open source projects: Apache, ArgoUML, and SVN. Compared with the SVM-54 scheme, the detection performance of SVM-SBCTC improved 2.79%–28.97% in the top-5 recall rates on three projects.

Tian et al. [36] extended the study of Jalbert and Weimer [20]. Duplicate bug reports were identified using relative similarity. It includes text similarity, surface features, and clustering. They considered several factors for this extension. The first was to use the extension of BM25 rather than using the term appearance count as similarity criteria. The information retrieval community widely use BM25, and it is more familiar than the term frequency (TF)-based similarity measure. BM25 is also known as the most accurate measure method of technical text searches [19]. Second, the "product" meta field of different bug reports was applied. Bug reports classified as different "product" are more likely to not be duplicated. Third, it is to build a hybrid function to examine the top N of the most similar bug reports, rather than using the best out of the most similar bug reports. Using the Mozilla project, Tian et al. [36] has demonstrated better performance than Jalbert and Weimer [20], improving the true positives (from 8% to 24%) and maintaining the false positives low (at 9%). Calculating the harmonic mean of the true positive rate and true negative rate improved the accuracy of the previous approach (from 14.8% to 38.6%).

3.5 Comparative Analysis

Fig. 5 shows the distribution of bug report deduplication studies based on the techniques. Table 1 shows the bug report deduplication study introduced in this study.

Most studies use the same experiment target such as a Mozilla, Eclipse, Firefox (a part of Mozilla), and Open Office project. All studies, except that of Hiew [32], are improvements over prior studies or have become a comparison target. There is a lack of research on studies of combined use of clustering and NLP techniques. Thus, such studies could be one of the open challenges in this field. In particular, most

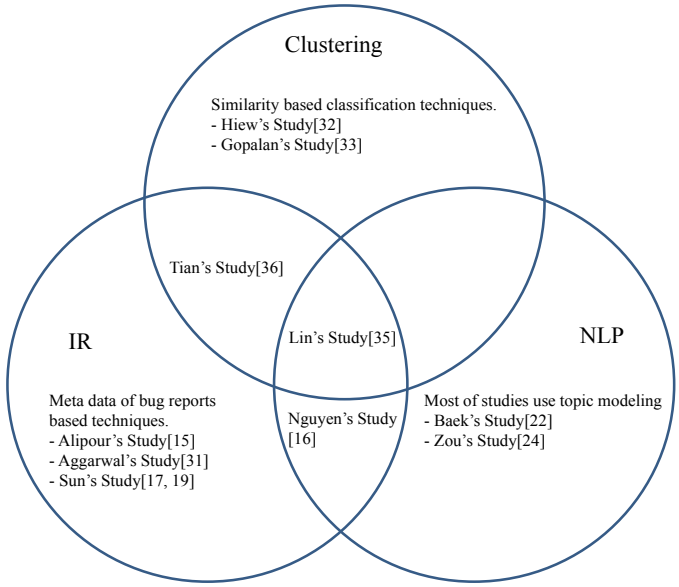

Fig. 5. Distribution of duplicate bug report identification techniques.

Table 1. Features of duplicate bug report identification techniques

Study	Used techniques	Experiment target (data set)	Compared target	Feature
Baek et al. [22]	NLP	Eclipse	Traditional machine learning techniques	Combine LDA, Naïve Bayes [23] and Naïve Bayes polynomial
Zou et al. [24], LNG	NLP	Eclipse	DBTM [16]	Combine LDA and a linearly coupled weight-based N-gram similarity.
Alipour et al. [15]	IR	Android	REP [19]	Apply contextual information to BM25F using pre-built software dictionaries
Aggarwal et al. [31]	IR	Android, Eclipse, Mozilla, Open Office	Alipour et al. [15]	Supplement Alipour et al. [15] and use software literature context instead of contextual information.
Sun et al. [19], REP	IR	Eclipse, Mozilla, Open Office	SVM	An extension of BM25F and uses non-textual fields (component, product, and version) as well as text content (summary and contents)
Sun et al. [17]	IR	Eclipse, Firefox, Open Office	Jalbert and Weimer [20], Runeson et al. [18], Wang et al. [37]	A discriminative model for searching similar bug reports in bug tracking systems.
Hiew [32]	Clustering	Eclipse, Firefox, Fedora, Apache	-	Base on topical detection and tracking techniques that considered clustering in news articles.
Gopalan and Krishna [33]	Clustering	Eclipse, Mozilla, Open Office	Jalbert and Weimer [20], Tian et al. [36]	A clustering-based technique to identify redundancy in a set of large bug reports.
Nguyen et al. [16], DBTM	Hybrid (NLP, IR)	Eclipse, Mozilla, Open Office	REP [19], RTM	Combine BM25F and T-Model, an extension of the Topic Modeling technique LDA.
Lin et al. [35], SVM-SBCTC	Hybrid (NLP, IR, clustering)	Apache, ArgoUML, SVN	Sun et al. [17]	SVM that considers the semantic correlation of text based on SVM discriminative scheme
Tian et al. [36]	Hybrid (IR, clustering)	Mozilla	Jalbert and Weimer [20]	Include text similarity, surface features, and clustering

clustering-based studies are compared with the representative work (e.g., [20]) or are proposed without any comparative analysis with existing techniques. Thus, a variety of verification schemes need to be considered. In addition to the technical aspects, bug report processing processes or frameworks could also be a research issue. There is also a need for studies on methodologies that effectively build and utilize these frameworks across the enterprise, using the various existing above-mentioned techniques.

4. Techniques for Improving Bug Triage Performance

It is generally impossible for a limited number of developers to process all the bug reports submitted every day, which are usually great in number. Developers usually fix bugs that are critical to the operation of the software or those that are more prioritized; this classification of bug reports by priority or severity is referred to as triage. Techniques to improve the performance of bug reports triage are mainly performed through machine learning.

4.1 Classification Algorithm

Numerous bug tracking systems use classification techniques, such as SVM [38,39] and KNN [40], based on machine learning. However, they are rarely used alone, and they are mostly integrated with a classification technique based on NB [41-44] classification or heterogeneity.

Anvik and Murphy [45] proposed a technique to create recommenders that assist with a variety of decisions aimed at streamlining the development process using machine learning techniques such as NB, EM [46,47], SVM, C4.5 (decision trees), nearest neighbor, and conjunctive rules. This technique extracts the normalized TF using the title and description. To evaluate this technique, they used bug reports from the Eclipse and Firefox projects. This technique showed 60% precision and 3% recall in Eclipse and 51% precision and 24% recall in Firefox.

Bhattacharya and Neamtiu [48] proposed a method using refined classification to improve bug report triage accuracy and reduce the length of the tossing paths. It extracts features such as TF-IDF or bag-of-words (BOW) using the NB and a tossing graph. It uses information such as bug report meta data of various types, title, description, keywords, product, component, and the last developer activities. They used Eclipse and Mozilla's bug reports to evaluate this technique and achieved 77.43% accuracy for Eclipse and 77.87% accuracy for Mozilla.

Kanwal and Maqbool [49] defined the features of bug reports and proposed a prioritized recommender based on the classifier. They compared the SVM with the NB algorithm, the most commonly used classification algorithm, to indicate the differences in performance by the classifier used in this technique. As a result of evaluating Eclipse projects, the SVM was found to be superior to the NB algorithm for text features, whereas for categorical features, the performance of NB was found to be better than that of the SVM. The highest accuracy is achieved with SVM when categorical and text features are combined for training.

Peng et al. [50] proposed a method to build a developer recommendation system to assign bugs based on relevant search scheme. This method consists of the index for bugs and searches for new bugs from the index. Then, it analyzes relevant bug and recommends it to the performance developer. They evaluated this method in Mozilla and Eclipse environments, and showed that it was better than machine

learning algorithms such as NB and SVM.

Xuan et al. [51] proposed techniques using SVM, NB for various factors such as developer prioritization and severity identification. This technique extracts TF-IDF and developer priorities using meta data such as titles and descriptions. The evaluation was conducted in the Eclipse and Mozilla bug reports and the technique was found to be superior to SVM or NB.

Xuan et al. [52] addressed the bug triage data reduction and improving the quality of bug data by reducing the scale. This method combines instance two selections: instance and feature. The method also reduces of bug- and word-dimensions at the same time by using the NB algorithm. This method also extracted TF from bug reports using title and description as meta data. It was evaluated using the bug reports of Eclipse and Mozilla. As a result, it was found to have an improved accuracy compared with the SVM, KNN, and NB.

Yang and Lee [53] proposed a method to predict the severity of a newly submitted bug report. When a new bug report is submitted, it finds a similar topic and uses the bug report's meta-fields to decrease the scope of the candidate bug report. It predicts the severity of the new bug report by training the extracted bug report with NB multinomial technique. They indicated that the method was more effective in predicting bug severity than NB, NB Multinomial, and KNN in Eclipse and Mozilla open-source projects. Its performance depends on the quality of the bug report. Generally, it is difficult to predict the bug reports as the Blocker label, because there are small number of bug reports identified as the Blocker label and many factors to be considered to classify the bug reports presented by various programming languages [15]. However, this study showed good predictions for the bug reports with Blocker severity level.

Yang et al. [54] proposed a new approach to predict the severity of bug reports based on emotional similarity. This approach uses a unigram model to identify emotional words, and searches for bug reports with emotional words using Kullback–Leibler divergence. They proposed emotion simplicity (ES)-Multinomial, a new algorithm that replaces the NB Multinomial. To compare ES-Multinomial with the existing NB Multinomial, they applied ES-Multinomial to open sources from Eclipse, GNU, JBoss, Mozilla, and WireShark and demonstrated that it was more efficient than the NB Multinomial.

Zhang et al. [55] proposed a technique called KNN search and heterogeneous proximity (KSAP) that uses the heterogeneous network of bug repository and historical bug reports to improve auto-allocation of bug reports. The KSAP is a two-step process. The first step is to search for similar historically fixed bug reports, and the second is to rank the developers who contributed to similar bug reports by heterogeneous proximity. They used projects from Mozilla, Eclipse, Apache Ant, and Apache Tomcat6 for evaluation, and there was a 7.5%–32.25% recall improvement compared with ML-KNN [56,57], DREX [58], DRETOM [59], Bugzie [60], DevRec [57], and developer prioritization (DP) method [51]. They showed that the KSAP was better than other modern techniques when the developer works less. Adjusting the number of similar historically fixed bug reports (K) and developers (Qs) maintains the superiority of KSAP.

Zhang et al. [61] proposed a severity prediction accuracy improvement method for automated techniques to replace manually assigned fixers. This method uses the REP [19] algorithm and KNN classification to find historical bug reports having the same features with input bugs. Then, it extracts the features to estimate the severity and recommend fixers. The method was applied to GCC, Open Office, Eclipse, NetBeans, and Mozilla's open source for evaluation and improvements in precision, recall, and F-measure were demonstrated compared with DRETOM, DREX, and DevRec.

4.2 Specialized Algorithm

Artificial neural networks, also used for deep learning, are often used in bug report triage. Conventional classification models such as BOW [62-64] are also replaced by techniques such as CNN and RNN.

Mani et al. [65] proposed a bug report representation algorithm that learns syntax and meaning in an unsupervised way using DBRNN-A (an agent-based stay secondary network). They compared DBRNN-A with cosine distance, NB, SVM, softmax classifier, and BOW model and showed that DBRNN-A provides a higher rank-10 average accuracy.

Most of the studies that use topic modeling and information retrieval techniques for triaging focus on the meta-data in bug reports. They use the NB Multinomial to supplement the technique or use information such as time and emotion to find ways to allocate better-performing bug reports.

Badashian et al. [66] proposed a new method to use Q&A community platforms such as GitHub, which is a source of developer expertise for bug triage. This method extracts keywords from bug reports using keywords (of metadata), project languages, tags from "GitHub" or "Stack overflow" as well as titles and descriptions. These keywords match social expertise. They evaluated this method using bug reports from 20 GitHub projects and showed an 89.43% accuracy.

Jonsson et al. [67] proposed a method using stacked generalization (SG) [68] as an ensemble learner to improve predictive accuracy in automatic bug allocation. SG is a state-of-the-art method that combines the output of various classifications used in various applications. One notable example is that SG-based solutions overwhelm competition in predicting movie ratings. In the field of software engineering, SG was applied in the prediction of the number of residual defects in the Black Box test [69] and detection of malware in smartphones [70]. The titles and descriptions were used for TF extraction and approximately 35,000 industrial project bug reports were used for evaluation. The results showed an 89% percent accuracy.

Shokripour et al. [71] presented an ABA-time-TF-IDF method, an auto-assignment technique based on TF-IDF time metadata. A corpus is constructed using nouns, and specialized knowledge is identified and recommended to developers. Shokripour compared the ABA-time-TF-IDF with ABA-TF-IDF, NB, VSM, SUM, and SVM in Eclipse, NetBean, and ArgoUML projects. As a result, accuracy and mean reciprocal rank (MBR) improved by up to 11.8% and 8.94%.

Another study by Shokripour et al. [72] proposed a two-phased method using assignment recommendations based on the predicted location of the bug. It addressed several problems of the activity-based method. This method uses source code information as well as title and description, where meaningful words are extracted. They used bug reports from Eclipse and Firefox to achieve an 89.41% accuracy and 59.76% accuracy for the assessment. However, the number of bug reports is smaller than that of other studies.

Tamrawi et al. [60] proposed Bugzie that recommends bug reports to developers by building a fuzzy-set based on words extracted from titles and descriptions. Bugzie greatly improved both accuracy and time efficiency compared with most conventional machine learning techniques such as NB, Bayesian Network, C4.5, SVM, incremental NB, and incremental Bayesian Network in the Eclipse bug report.

Wang et al. [73] proposed FixerCache, an unsupervised bug triage technique. FixerCache has overcome the problems of supervised classification based on the activation of its product components. FixerCache uses active developer caches to extract TF from the title and description of the bug. They evaluated FixerCache in a bug report from Eclipse and Mozilla, showing better accuracy than the SVM and NB.

Wen et al. [74] proposed the Configuration Bug Learner Uncovers Approved options (CoLUA), a two-step automation technique that integrates NLP, IR, and machine learning, to address communication problems between bug reporters and developers. As the first step, CoLUA selects functions in the text information of the bug report and applies machine learning techniques to create a triage model. The second step identifies the configuration options that are included in the labeled bug report. CoLUA was applied to open-source projects from Mozilla, Apache, and MySQL. As a result, the average F-measure for the ZeroR classifier was found to be 0.33, whereas the average F-measure for CoLUA is 0.73 for all three projects.

4.3 Comparative Analysis

Table 2 lists the bug report triage studies that are introduced in this study. Most studies use the same experiment target such as Mozilla, Eclipse, Firefox (a part of Mozilla), and Open Office project. In addition, the influence of the NB and SVM methods was very strong in bug triage, i.e., 13 out of 17 studies used the NB or SVM methods. Thus, in order to improve the performance of bug report triage, it is necessary to carry out studies using various techniques, such as clustering, deep learning, and graph theory. In particular, deep learning-based approaches could be used as a key option to provide potential synergy with existing techniques. In addition to the technical aspects, enterprise-wide frameworks for improving efficiency of the bug report triage must be studied.

Table 2. Features of bug report triage techniques

Study	Used techniques	Experimental target (data set)	Compared target	Feature
Anvik and Murphy [45]	NB, EM, SVM, C4.5, nearest neighbor, conjunctive rules	Eclipse, Firefox	-	Create recommenders that assist with a variety of decisions aimed at streamlining the development process.
Bhattachira and Neamtiu [48]	NB, Tossing graph	Eclipse, Mozilla	NB, Bayesian Network	Use refined classification and reduce the length of the tossing paths.
Kanwal and Maqbool [49]	SVM, NB	Eclipse	SVM, NB	Define the features of bug reports and proposes a prioritized recommender based on the classifier.
Peng et al. [50]	Indexing for searching relevant bug reports	Eclipse, Mozilla	SVM, NB	Recommend developers to assign bugs based on relevant search
Xuan et al. [51]	SVM, NB	Eclipse, Mozilla	SVM, NB	Use SVM, NB for developer prioritization, severity identification, etc.
Xuan et al. [52]	NB	Eclipse, Mozilla	SVM, NB, KNN	Address the bug triage data reduction and how to improve the quality of bug data by reducing the scale.
Yang and Lee [53]	Naïve Bayes Multinomial	Eclipse, Mozilla	NB, Naïve Bayes Multinomial, KNN	Predict the severity of the new bug report by training the extracted bug report with Naïve Bayes Multinomial technique.
Yang and Lee [54], Simplicity-Multinomial	Naïve Bayes Multinomial, Kullback-Leibler divergence	Eclipse, Mozilla, JBoss, WireShark	Naïve Bayes Multinomial	Predict the severity of bug reports based on emotional similarity.

(Continued on the next page)

Table 2. Continued

Study	Used techniques	Experimental target (data set)	Compared target	Feature
Zhang et al. [55], KSAP	Heterogeneous network	Eclipse, Mozilla, Apache Ant, Apache Tomcat6	DREX [58], DRETOM [59], Bugzei [60], DevReg [55], developer prioritization [51]	Use heterogeneous network of bug repository and historical bug reports
Zhang et al. [61]	REP [19], KNN	Eclipse, Mozilla, Open Office, GCC, NetBeans	DREX [58], DRETOM [59], DevReg [57]	Use the REP [19] algorithm and KNN classification to find historical bug reports similar to new bugs.
Mani et al. [65]	Deep learning	Google Chromium, Core, Fire Fox	BOW model, softmax classifier, SVM, NB, cosine distance	Build model that learns a syntactic and semantic feature
Badashian et al. [66]	Keywords matching	Github projects	-	Use Q&A community platforms, which is a source of developer expertise for bug triage
Jonsson et al. [67]	Stacked generalization [68]	Industrial projects	-	Use stacked generalization [68] as an ensemble learner.
Shokripour et al. [71], ABA-time-TF-IDF	TF-IDF	Eclipse, NetBean, ArgoUML	ABA-TF-IDF, NB, VSM, SUM, SVM	Propose an auto-assignment technique based on TF-IDF's time metadata.
Shokripour et al. [72]	Words extraction	Eclipse, Firefox	-	Propose a two-phase method using assignment recommendations based on the predicted location of the bug, and address several problems of the activity-based method.
Tamrawi et al. [60], Bugzie	Fuzzy set	Eclipse	NB, Bayesian Network, C4.5, SVM, incremental Naïve Bayes, incremental Bayesian Network	Recommend bug reports to developers by building a fuzzy set.
Wang et al. [73], FixerCache	Unsupervised training	Eclipse, Mozilla	SVM, NB	Overcome the problems of supervised classification based on the activation of its product components
Wen et al. [74], CoLUA	NLP, IR, CBR	Mozilla, Apache MySQL	ZeroR	A two-step automation technique that address communication problems between bug reports and developers.

5. Conclusion

This study systematically addressed the bug report processing techniques for improving software management. A number of studies were discussed based on identification of duplicate bug reports and triage bug reports. The studies were classified for the purpose of discussion, such as used techniques, experiment target, and compared target. The results indicated the future research direction of bug report processing techniques.

Software bugs are inevitable during the development and maintenance phases. Because it is specified and managed in the form of software bug reports, it is necessary to study more efficient bug report processing techniques. Through this survey study, in our viewpoint, there are further issues in the two mainstreams of bug report studies: reducing duplicate bug reports and improving bug triage performance. From a technical point of view, one of the core techniques is identifying the similarity between bug reports

in this research field. Thus, it is necessary to study this technique in detail to build elaborate models for the two mainstreams issues mentioned above. In particular, deep learning-based and graph-based approaches [75-77] are in their early stages, and could be one option. There are still many deep learning algorithms and graph theories that can be applied to bug report processing systems. As more elaborate models are proposed in this field, the efficiency of the bug processing system can be increased, and consequently software development effort can be reduced. Thus, the researchers and practitioners must focus on progress of similarity studies between bug reports. From a research methodological point of view, in order to validate the superiority of the proposed techniques, it is necessary to analyze it with various existing techniques for efficient bug report processing. Instead of analyzing only the proposed technique itself, meaningful and practical results through comparative analysis with the various techniques proposed in the past showed be provided. Through such studies, it would be of great significance if software developers can be guided on which techniques should be used for reducing duplicate bug reports or improving bug triage performance in their software development environments.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1C1B5018295).

References

[1] S. Just, R. Premraj, and T. Zimmermann, "Towards the next generation of bug tracking systems," in *Proceedings of 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, Herrsching am Ammersee, Germany, 2008, pp. 82-85.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful… really?," in *Proceedings of 2008 IEEE International Conference on Software Maintenance*, Beijing, China, 2008, pp. 337-345.

[3] T. Dal Sasso and M. Lanza, "In∗ bug: visual analytics of bug repositories," in *Proceedings of 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, Antwerp, Belgium, pp. 415-419.

[4] A. Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M. T. Valente, and J. Martins, "Bug maps: a tool for the visual exploration and analysis of bugs," in *Proceedings of 2012 16th European Conference on Software Maintenance and Reengineering*, Szeged, Hungary, 2012, pp. 523-526.

[5] M. D'Ambros, "Supporting software evolution analysis with historical dependencies and defect information," in *Proceedings of 2008 IEEE International Conference on Software Maintenance*, Beijing, China, 2008, pp. 412-415.

[6] Life cycle of a bug [Online]. Available: https://www.bugzilla.org/docs/2.18/html/lifecycle.html.

[7] P. Knab, B. Fluri, H. C. Gall, and M. Pinzger, "Interactive views for analyzing problem reports," in *Proceedings of 2009 IEEE International Conference on Software Maintenance*, Edmonton, Canada, 2009, pp. 527-530.

[8] Y. Takama and T. Kurosawa, "Application of monitoring support visualization to bug tracking systems," in *Proceedings of 2013 IEEE International Symposium on Industrial Electronics*, Taipei, Taiwan, 2013, pp. 1-5.

[9] T. Dal Sasso, A. Mocci, and M. Lanza, "What makes a satisficing bug report?," in *Proceedings of 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Vienna, Austria, 2016, pp. 164-174.

[10] 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*, San Diego, CA, 2005, pp. 35-39.

[11] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, China, 2006, pp. 361-370.

[12] Bug report: BE-268 in JIRA [Online]. Available: https://jira.hyperledger.org/projects/BE/issues/BE-268?filter=allopenissues

[13] MantisBT, "Bug report: projects and category (not global categories)," 2013 [Online]. Available: https://mantisbt.org/bugs/view.php?id=15501.

[14] Bug report: Bug 1449567 in Bugzilla [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=1449567

[15] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proceedings of 2013 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA, 2013, pp. 183-192.

[16] 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 *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Essen, Germany, 2012, pp. 70-79.

[17] C. Sun, D. Lo, X. Wang, J. Jiang, and S. C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Cape Town, South Africa, 2010, pp. 45-54.

[18] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th International Conference on Software Engineering*, Minneapolis, MN, 2007, pp. 499-510.

[19] C. Sun, D. Lo, S. C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, Lawrence, KS, 2011, pp. 253-262.

[20] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proceedings of 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, Anchorage, AK, 2008, pp. 52-61.

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993-1022, 2003.

[22] S. S. Baek, G. S. Yang, J. W. Lee, and B. J. Lee, "Detecting duplicate bug reports for reducing developers' workload," in *Proceedings of the 2016 KISS Conference*, 2016, pp. 634-636.

[23] S. J. Dommati, R. Agrawal, R. M. Reddy G, and S. S. Kamath, "Bug classification: feature extraction and comparison of event model using naive Bayes approach," 2013 [Online]. Available: https://arxiv.org/abs/1304.1677.

[24] J. Zou, L. Xu, M. Yang, X. Zhang, J. Zeng, and S. Hirokawa, "Automated duplicate bug report detection using multi-factor analysis," *IEICE Transactions on Information and Systems*, vol. 99, no. 7, pp. 1762-1775, 2016.

[25] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," in *Proceedings of 2015 IEEE 23rd International Requirements Engineering Conference (RE)*, Ottawa, Canada, 2015, pp. 116-125.

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

[27] S. Banitaan and M. Alenezi, "Tram: an approach for assigning bug reports using their metadata," in *Proceedings of 2013 3rd International Conference on Communications and Information Technology (ICCIT)*, Beirut, Lebanon, 2013, pp. 215-219.

[28] V. Guana, F. Rocha, A. Hindle, and E. Stroulia, "Do the stars align? Multidimensional analysis of Android's layered architecture," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, Zurich, Switzerland, 2012, pp. 124-127.

[29] A. Hindle, N. A. Ernst, M. W. Godfrey, and J. Mylopoulos, "Automated topic naming to support cross-project analysis of software maintenance activities," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, Honolulu, HI, 2011, pp. 163-172.

[30] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding android fragmentation with topic analysis of vendor-specific bugs," in *Proceedings of 2012 19th Working Conference on Reverse Engineering*, Kingston, Canada, 2012, pp. 83-92.

[31] K. Aggarwal, F. Timbers, T. Rutgers, A. Hindle, E. Stroulia, and R. Greiner, "Detecting duplicate bug reports with software engineering domain knowledge," in *Proceedings of 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, Montreal, Canada, 2015, pp. 211-220.

[32] L. Hiew, "Assisted detection of duplicate bug reports," M.S. thesis, University of British Columbia, Vancouver, Canada, 2006.

[33] R. P. Gopalan and A. Krishna, "Duplicate bug report detection using clustering," in *Proceedings of 2014 23rd Australian Software Engineering Conference*, Milsons Point, Australia, 2014, pp. 104-109.

[34] J. Chang and D. Blei, "Relational topic models for document networks," in *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, Clearwater, FL, 2009, pp. 81-88.

[35] M. J. Lin, C. Z. Yang, C. Y. Lee, and C. C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *Journal of Systems and Software*, vol. 121, pp. 223-233, 2016.

[36] Y. Tian, C. Sun, and D. Lo, "Improved duplicate bug report identification," in *Proceedings of 2012 16th European Conference on Software Maintenance and Reengineering*, Szeged, Hungary, 2012, pp. 385-390.

[37] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei, "Extracting paraphrases of technical terms from noisy parallel software corpora," in *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, Singapore, 2009, pp. 197-200.

[38] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293-300, 1999.

[39] S. Tong and E. Chang, "Support vector machine active learning for image retrieval," in *Proceedings of the 9th ACM International Conference on Multimedia*, Ottawa, Canada, 2001, pp. 107-118.

[40] H. Zhang, A. C. Berg, M. Maire, and J. Malik, "SVM-KNN: discriminative nearest neighbor classification for visual category recognition," in *Proceedings of 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* New York, NY, 2006, pp. 2126-2136.

[41] A. McCallum and K. Nigam, "A comparison of event models for naive Bayes text classification," in *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*, Madison, WI, 1998, pp. 41-48.

[42] D. D. Lewis, "Naive (Bayes) at forty: the independence assumption in information retrieval," in *Machine Learning: ECML-98.* Heidelberg: Springer, 1998, pp. 4-15.

[43] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes," *Advances in Neural Information Processing Systems*, vol. 15, pp. 841-848, 2002.

[44] I. Rish, "An empirical study of the naive Bayes classifier," in *Proceedings of IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, Seattle, WA, 2001, pp. 41-46.

[45] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, article no. 10, 2011.

[46] A. McCallum, "Multi-label text classification with a mixture model trained by EM," 1999 [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.888.

[47] D. Cubranic and G. Murphy, "Automatic bug triage using text categorization," in *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering*, Banff, Canada, 2004, pp. 92-97.

[48] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Proceedings of 2010 IEEE International Conference on Software Maintenance*, Timisoara, Romania, 2010, pp. 1-10.

[49] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *Journal of Computer Science and Technology*, vol. 27, no. 2, pp. 397-412, 2012.

[50] X. Peng, P. Zhou, J. Liu, and X. Chen, "Improving bug triage with relevant search," in *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering*, Pittsburgh, PA, 2017, pp. 123-128.

[51] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proceedings of 2012 34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland, 2012, pp. 25-35.

[52] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 264-280, 2014.

[53] G. Yang and B. Lee, "Predicting bug severity by utilizing topic model and bug report meta-field," *KIISE Transactions on Computing Practices*, vol. 21, no. 9, pp. 616-621, 2015.

[54] G. Yang, T. Zhang, and B. Lee, "An emotion similarity based severity prediction of software bugs: a case study of open source projects," *IEICE Transactions on Information and Systems*, vol. 101, no. 8, pp. 2015-2026, 2018.

[55] W. Zhang, S. Wang, and O. Wang, "KSAP: an approach to bug report assignment using KNN search and heterogeneous proximity," *Information and Software Technology*, vol. 70, pp. 68-84, 2016.

[56] M. L. Zhang and Z. H. Zhou, "ML-KNN: a lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038-2048, 2007.

[57] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Proceedings of 2013 20th Working Conference on Reverse Engineering (WCRE)*, Koblenz, Germany, 2013, pp. 72-81.

[58] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "Drex: developer recommendation with k-nearest-neighbor search and expertise ranking," in *Proceedings of 2011 18th Asia-Pacific Software Engineering Conference*, Ho Chi Minh, Vietnam, 2011, pp. 389-396.

[59] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "Dretom: developer recommendation based on topic models for bug resolution," in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, Lund, Sweden, 2012, pp. 19-28.

[60] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, Szeged, Hungary, 2011, pp. 365-375.

[61] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166-184, 2016.

[62] H. M. Wallach, "Topic modeling: beyond bag-of-words," in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006, pp. 977-984.

[63] J. Yang, Y. G. Jiang, A. G. Hauptmann, and C. W. Ngo, "Evaluating bag-of-visual-words representations in scene classification," in *Proceedings of the International Workshop on Workshop on Multimedia Information Retrieval*, Augsburg, Germany, 2007, pp. 197-206.

[64] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in twitter to improve information filtering," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Geneva, Switzerland, 2010, pp. 841-842.

[65] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: exploring the effectiveness of deep learning for bug triaging," 2018 [Online]. Available: https://arxiv.org/abs/1801.01275.

[66] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," in *Proceedings of 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bremen, Germany, 2015, pp. 506-510.

[67] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1533-1578, 2016.

[68] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.
[69] N. Li, Z. Li, Y. Nie, X. Sun, and X. Li, "Predicting software black-box defects using stacked generalization," in *Proceedings of 2011 6th International Conference on Digital Information Management*, Melbourn, Australia, 2011, pp. 294-299.
[70] A. Amamra, C. Talhi, J. M. Robert, and M. Hamiche, "Enhancing smartphone malware detection performance by applying machine learning hybrid classifiers," in *Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity*. Heidelberg: Springer, 2012, pp. 131-137.
[71] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "A time-based approach to automatic bug report assignment," *Journal of Systems and Software*, vol. 102, pp. 109-122, 2015.
[72] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of 2013 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA, 2013, pp. 2-11.
[73] S. Wang, W. Zhang, and Q. Wang, "FixerCache: unsupervised caching active developers for diverse bug triage," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014.
[74] W. Wen, T. Yu, and J. H. Hayes, "Colua: automatically predicting configuration bug reports and extracting configuration options," in *Proceedings of 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Ottawa, Canada, 2016, pp. 150-161.
[75] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proceedings of 2012 34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland, 2012, pp. 419-429.
[76] G. Erkan and D. R. Radev, "Lexrank: graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457-479, 2004.
[77] P. Liu, J. Wang, A. K. Sangaiah, Y. Xie, and X. Yin, "Analysis and Prediction of Water Quality Using LSTM Deep Neural Networks in IoT Environment," *Sustainability*, vol. 11, no. 7, 2058, 2019.

Dong-Gun Lee https://orcid.org/0000-0001-6792-4572

He received the B.S. degree in computer engineering from Yeungnam University, Korea, in 2019. He is currently a M.S. student in the Department of Computer Engineering, Yeungnam University, Korea. His research interests include software engineering, open source software, software defect prediction, and edge computing.

Yeong-Seok Seo https://orcid.org/0000-0002-5319-7674

He received the B.S. degree in computer science from Soongsil University, Korea, in 2006, and the M.S. and Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2008 and 2012, respectively. From September 2012 to December 2013, he was a postdoctoral researcher in KAIST Institute for Information and Electronics. From January 2014 to August 2016, he was a senior researcher in Korea Testing Laboratory (KTL), Korea. He is currently an assistant professor in the Department of Computer Engineering, Yeungnam University, Korea. His research interests include software engineering, artificial intelligence, Internet of Things (IoT), and data mining.